 Proceedings of 4th EARSeL Workshop on Imaging Spectroscopy. New quality in environmental studies. Zagajewski B., Sobczak M., Wrzesień M., (eds)

PARALLELIZATION OF APEX AIRBORNE IMAGING SPECTROMETER PRODUCT GENERATION

Jason Brazile[1], Johannes W. Kaiser[1], Daniel Schläpfer[1], Jens Nieke[1], Michael E. Schaepman[2], and Klaus I. Itten[1]

1. Remote Sensing Labs, Department of Geography, University of Zurich, Zurich, Switzerland, Jason.Brazile@geo.unizh.ch
2. Centre for Geo-Information and Remote Sensing, Wageningen University, Wageningen, The Netherlands

ABSTRACT

The APEX airborne imaging spectrometer [i] is designed to simultaneously acquire up to 511 VNIR/SWIR un-binned channels of 1000 16-bit across track pixels with typical flight lines containing tens of hundreds of lines and typical scenes composed of multiple flight lines. The on-board computer is capable of recording up to 300 GB of raw image data. [ii]

The instrument is currently in hardware construction and integration phase and therefore hasn't yet been fully characterized, but based on expected instrument performance and specifications, a forward instrument model has been defined along with its corresponding inverse processing model for producing at-sensor radiance from raw digital numbers.

In the worst case, this processing is predicted to require up to three days of CPU time on a single processor system. However, due to independent per-frame processing, use of additional spare workstations could reduce this to less than one day, with scalability limited mostly by network input/output bandwidth.

The experimental parallelization of the currently implemented APEX processor is described including run-time analysis, parallelization strategy, estimations of speedup, and discussion of selected implementation issues.

INTRODUCTION

As a background to the problem, a high-level view of the data flow and resulting products is given followed by a summary of the instrument model and corresponding inverse model processing used to produce the primary level 1 product. Finally this information motivates the run-time analysis and parallelization strategy.

APEX Data Products

The specification and definition of APEX data products follows the typical numerical conventions of hyperspectral imagery and uses a trailing letter to distinguish between optional variations:

Table 1: APEX data products

Level 0B	Raw data segregated by run into well-defined file formats
Level 1A	At-sensor radiance - no re-sampling or smoothing
Level 1B	At-sensor radiance - uniformity only per detector in 2 spectral regions
Level 1C	At-sensor radiance - full spatial/spectral uniformity
Level 2	Surface reflectance

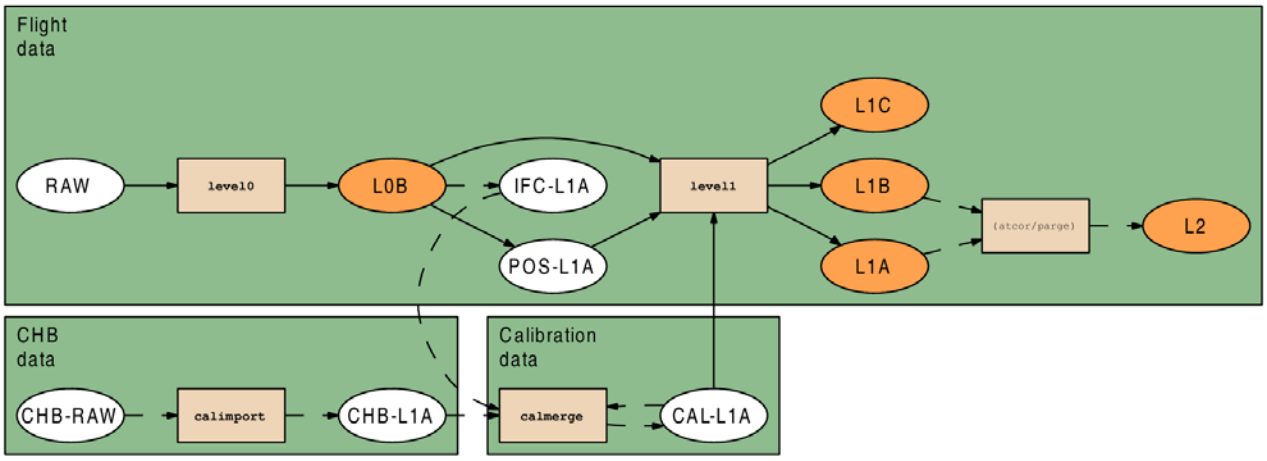

Figure 1: APEX data products

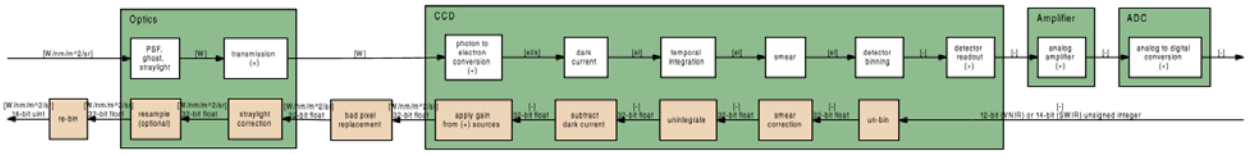

Figure 2: Forward instrument model (top) and inverse processing model (bottom)

The differences among the level 1 offerings are due to uniformity issues between the two spectral channels. Users who wish to use at-sensor radiance will probably desire full spatial/spectral uniformity among the two sensors while other users may wish direct access to full spatial/spectral resolution even in the presence of intra-sensor non-uniformity because e.g. further higher level processing is desired.

Full descriptions and definitions of the APEX data products are described elsewhere [iii], [iv] but a graphical summary of the processing work flow and their resulting products is provided in Figure 1.

The APEX Instrument Model

The APEX instrument is currently in its hardware construction and integration phase and therefore its characteristics are still based primarily on design requirements and estimates derived from bread-board test measurements on selected components provided by the industrial integration team. However the expected range of anomalies and their potential affects have been modeled and analyzed [v] resulting in a well-defined forward instrument model [iv] and corresponding inverse processing model, which is summarized in Figure 2.

Parallelization Strategy

Because processing is performed on a frame-by-frame basis with no interdependencies, there is no data flow or scheduling specific constraints imposed on how the problem should be decomposed. Such “embarrassingly parallel” problems can be solved somewhat generically - two possible approaches being a task-centric manner where certain sub-tasks are assigned to individual processors or a data-centric manner where certain subsets of the data are assigned to individual processors. Due to uncertainty of the final runtime proportion of sub-tasks (i.e. additional time-consuming corrections may be needed once real data is available), a simple data centric model was chosen for APEX level 1 processing. A summary of the chosen parallelization strategy is therefore as follows:

1. Develop a run-time processing model
2. Profile real processing time using simulated data to apply model
3. Estimate theoretical limits based on model and profiling
4. Develop parallelization/decomposition plan based on frame-by-frame processing
5. Use model to estimate results and speedup

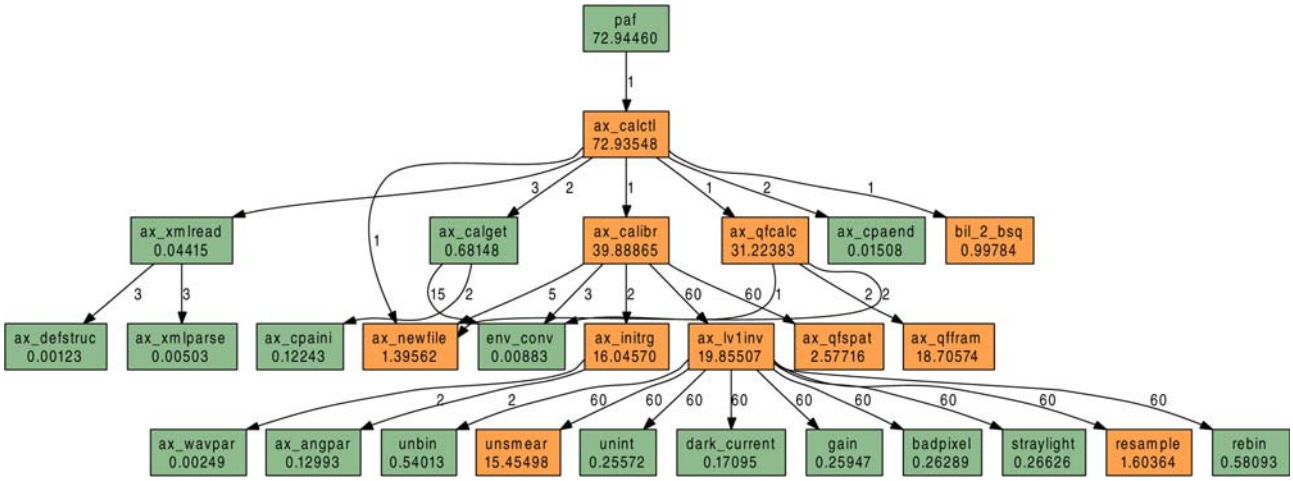

Figure 3: Call graph of level 1B product generation on simulated 30 frame data set (boxes labeled with time in seconds, lines with number of times called)

ANALYSIS AND DISCUSSION

Runtime processing model

In typical high volume data applications, the cost of input/output is expected to contribute a measurable and sometimes substantial percentage of run-time cost. Therefore, run-time processing for the single CPU case has been modeled as follows:

$$T_{total} = T_{input} + T_{processing} + T_{output}$$

In the parallel case, additional terms are needed for scattering the input among the different CPUs as well as gathering their results, and the processing term is replaced by the runtime of the slowest running of the independent T_i jobs:

$$T_{total} = T_{input} + T_{scatter} + \max(T_i) + T_{gather} + T_{output}$$

Since it was decided that jobs would correspond to frames or sets of frames, many more jobs than CPUs are expected, so an attempt is made to reduce variation among T_i runtime terms (due to differing CPU speeds in an assumed heterogeneous cluster or grid) by dynamically scheduling jobs to CPUs in a self-regulating way (further described below). Changing the model to reflect this is not required for making lower limit run-time predictions.

Common Case (Level 1B) Profiling

A pre-release of the APEX operational processor has been developed which performs the **`level0`** and **`level1`** processing steps as shown in Figure 1. A small synthetic test data set of 30 frames was developed for functional and numeric testing of the processor. As this is the most realistic data set available, it is also used for run-time profiling. Since level 2 processing is out of scope of the current APEX C/D development phase, the dominant processing time for standard product generation is therefore taken by level 1 processing. A run-time call graph was produced by profiling the

APEX processor while generating the 1B product from the test data set using a commodity Linux workstation based on the AMD Athlon 64 3200+ processor (see Figure 3).

From the generated call graph, it can be seen that the initialization (**`ax_initrg`**) and quality flag determination (**`ax_qfcalc`**/**`ax_qffram`**) routines take a substantial percentage of the total running time, however this is only true because the data set is so small. These routines are called only before starting and after finishing frame-by-frame processing hence their relative time approaches zero with more realistic-sized data sets.

The point of entry for frame-by-frame processing is **`ax_calibr`** which calls **`ax_lv1inv`** twice for each frame (once for the VNIR region, and once for the SWIR) and applies the inverse model transformations as defined by the instrument model (Figure 2). The run-time is therefore dominated by smear correction, which is essentially calling IDL's **`cholsol`** Cholesky solver and therefore is not open to micro-optimization. The next routine to watch is **`resample`** which is primarily IDL's **`interpolate`** function, which defaults to bilinear interpolation.

Since real instrument data is not yet available, it is not known if the current implementations of smear correction or re-sampling can be algorithmically simplified on the one hand or need to be made more sophisticated on the other.

Estimating theoretical Limits

In the run-time models defined above, the input/output and scatter/gather terms are invariant relative to the number of CPUs. This is not true in the latter case when using commodity networking hardware. However, these terms in the model can still be used to provide an absolute lower bound on estimated run time - regardless of how much speed-up parallelization may achieve, overall run time will never be less than the time taken for the input/output and scatter/gather terms.

It is possible to easily measure approximations of these parameters with standard operating system commands. The **`dd`** and **`cp`** commands can be used for approximations of T_{output} and $T_{input} + T_{output}$ and piping through **`ssh`** can approximate $T_{scatter} + T_{gather}$. However since an **`ssh`** pipeline includes encryption computation and data copying overhead, an additional estimate was made using Spinellis' non-standard **`socketpipe`** utility [vi], which gives a better estimate of possible application bandwidth. A sample data size greater than the size of RAM was needed as well as the **`sync`** command to ensure that complete disk reading/writing was being measured and not merely operating system cache or buffer throughput operations. The following results were obtained measuring between two identical commodity AMD Athlon 64 3200+ based Linux workstations with built-in gigabit ethernet (using a crossover cable). The only “optimization” made was that input was read from one disk while output was written to another (but both disks in the same machine).

Table 2: Measured and extrapolated run-time model parameters

Approximated Parameter	Measured (3GB) (4792 frames)	Throughput	Extrapolated (300GB)
T_{output} [1]	00:02:18	~22 MB/s	03:50:00
$T_{input} + T_{output}$ [2]	00:02:21	~22 MB/s	03:55:00
$T_{scatter} + T_{gather}$ [3]	00:04:51	~10 MB/s	08:05:00
$T'_{scatter} + T'_{gather}$ [4]	00:01:30	~33 MB/s	02:30:00
$T_{input} + T_{proces\,sing}$ [5]	00:52:37	~1MB/s	87:41:40

Applying the above measured parameters to the parallel run-time model implies that disk and network input/output for a full 300 GB input on current commodity hardware will run no faster than about six and a half hours. However, it is predicted that filesystem write performance could probably be doubled by using multiple disks configured as a single RAID device thereby reducing this to about five and half hours.

However, the above measurements also show that the CPU term of overall processing time reveals high potential for parallelization. If perfect scalability is assumed, only four compute nodes could reduce overall processing runtime to a single day in the worst case of 300 GB input. As previously mentioned, “embarrassingly parallel” problems such as these are the most likely kinds of problems to approach perfect scalability. In this case, a key implementation issue toward achieving this will be overlapping scatter/gather network input/output with per node CPU processing. Given the large difference in throughput between processing and networking as shown by the above measurements, this should be feasible.

Parallelization/Decomposition Plan

The previously discussed analysis and estimates led to the following design strategy:

> **Maximize filesystem write bandwidth on the system doing the final merge.** Because the final write is expected to be the largest single I/O cost, effort expended to improve filesystem write bandwidth (e.g. via an inexpensive software RAID on a single node) could have a significant impact on total processing time.
>
> **No disk I/O on compute nodes.** Jobs allocated to compute nodes should be sized such that everything can be computed in memory. With typical 300 band binning, a compute

[1] sync && time sh –c ‘dd if=/dev/zero of=/d1/z bs=100000 count=30000 && sync’

[2] sync && time sh –c ‘cp /d1/z /d2 && sync’

[3] time dd if=/dev/zero bs=100000 count=30000 | ssh node2 cat | dd bs=100000 of=/dev/null

[4] time socketpipe –l { dd if=/dev/zero bs=100000 count=30000 } –l {ssh node2 } –r { cat } –o { dd bs=100000 of=/dev/null }

[5] time sh –c ‘idl –rt=level1b_stream_test.sav < input_frames_4792 > /dev/null’

node with 512MB of RAM should be able to process and buffer results from on the order of 400 frames without any disk I/O. Such buffering would also support simultaneous network and CPU processing.

Decompose processing by independent sets of frames. Since it is not known if the current APEX processor implementation contains all necessary or sufficient corrections, this simple decomposition is the most flexible for future maintenance.

Efficiently schedule subtasks to (heterogeneous) compute nodes. Since spare resources typically have varying CPU and I/O capabilities, a scheduling scheme that allocates jobs to compute nodes in proportion to their capabilities is worthwhile.

Since the decomposed sub tasks are independent from one another, a parallel implementation could just as easily be mapped to a cluster as a grid - each having their respective advantages and disadvantages [vii]. To minimize intrusive code changes and maximize deployment possibilities, a grid implementation [viii] with the following design was chosen for this investigation:

1. One compute node is dedicated to input/output
2. The I/O node splits input into jobs
3. Job server on I/O node asynchronously serves new jobs to compute nodes
4. Compute nodes serially request jobs from the job server on the I/O node
5. The I/O node asynchronously merges results as they arrive from compute nodes

A visualization of a possible execution of this system is shown in Figure4. This example assumes four nodes - one dedicated to input/output and three used as compute nodes. Node 3 is assumed to have a processing throughput of only half that of Node 1, while Node 2 is in between. Differences in processing throughput could be due to factors such as CPU speed or network bandwidth. If there are only 3 jobs (e.g. total number of frames are equally divided) the total run time is limited by the slowest compute node and the remaining compute nodes could be mostly idle. If there are several jobs (e.g. each frame is job), faster compute nodes will efficiently process a larger percentage of them. However, having too many jobs increases the potential for slowing due to network latency, therefore job partitioning should be bounded above by the number that can be processed completely in memory and bounded below by network message latency.

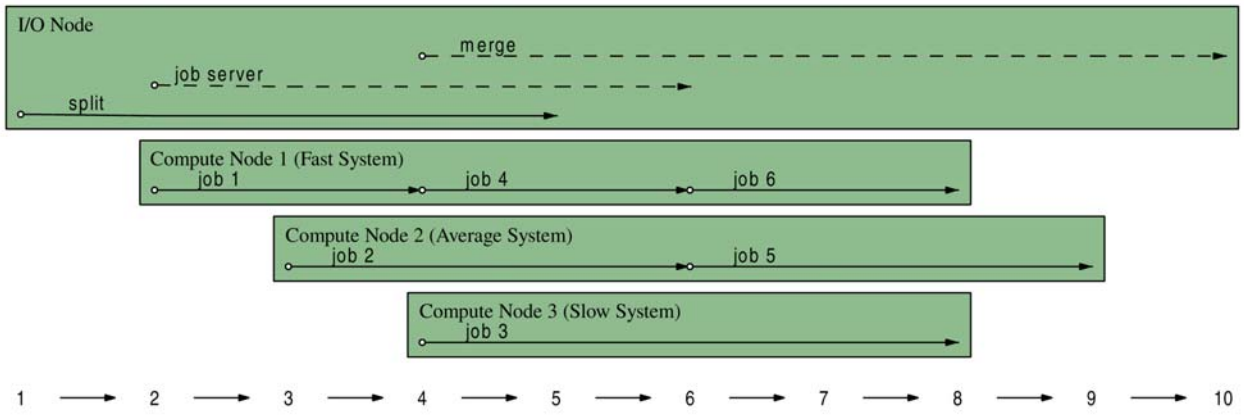

Figure 4: Possible execution time line (arbitrary scale) of a six job run (e.g. five frames per job for given 30 frame sequence) on four heterogeneous nodes

RESULTS AND STATUS

The APEX airborne imaging spectrometer is undergoing construction and system integration [i]. At a high level, the instrument's products have been well-defined and at a low level, an instrument model has been defined and a corresponding inverse model data processor has been implemented for producing at-sensor radiance [iii], [iv]. This processor has been run against simulated data based on the instrument's currently known characteristics - the unknowns provided by specifications.

For operational processing of the expected high-volume data, analysis and design for a parallelization of the data processor has been performed. The original serial data processor was modified to allow stream processing of individual frames and simulated test data was used to provide parameters to a run-time performance model allowing lower-bound estimation of parallel run-time performance. An open source generic grid framework has been developed for proposed deployment of the operational processor [viii]. Completing the operational parallel version of the processor will require integration of the stream processor into the chosen grid framework. However, this is expected to require minimal development resources. It is predicted that four spare compute nodes (e.g. two dual-CPU workstations) would be sufficient for reducing the estimated processing time of the 300 GB input data worst case from three days to within the stated goal of a single day.

OUTLOOK

If the modifications made for stream processing were carried to their logical conclusion and the processor were split into separate standalone components operated as filters, data processing could be theoretically configured at runtime according to a typical UNIX pipeline:

```
unbin frames | unsmear | unint | darkcurrent | gain | badpix | resample | rebin > cube-rad
```

This would enable desirable features such as the following:

- Data converters could be pre-pended/appended

  `bsq2bil frames-bsq | ...`

  `... | to_envi > ...`

- “Quirks” for different instruments could be encapsulated (and contributed by others)

  `... | hyperion_destripe | hyperion_rotate 2.1 | ...`

- A generic GUI for feedback could be placed at any stage in the pipeline

  `... | progress_bar | ...`

- Since each stage is standalone, any stage could be written in any computer language, increasing the chances of external contributions.
- Multiple interchangeable implementations of the same stage could be chosen at run-time e.g. **`isdas_smile_detect`** vs **`gao_smile_detect`**.
- It would be possible to mix and match binary-only proprietary stages (perhaps for intellectual property reasons) with open source stages.
- Task-level parallelism could be dynamically introduced via tools such as **`ssh`** or **`socketpipe`**

  `... | ssh node1 stage1 | ssh node2 stage2 | ...`

One issue to overcome would be the need for supporting multiple inputs (e.g. in addition to image data, metadata also needs to pass from stage to stage) while still allowing streaming of the primary image data. However, it is theorized that this could be accommodated by using a streaming archive manipulation library. For example, the Java programming environment implements such a library and programming interface for manipulating .jar files in .ZIP archive format. However, more generic and higher performance libraries are also available such as Kientzle's **`libarchive`** [ix]

which has already been used to solve a similar stream + metadata problem in implementing the **pkg_add** [x] package management tool. If these kinds of issues can be overcome, the resulting increase in processing flexibility would likely outweigh the decrease in efficiency due to data copying between pipeline stages - especially if parallelization of these pipelines reduce running times to acceptable levels of performance.

ACKNOWLEDGMENTS

This work was supported in part by ESA/ESTEC contracts 16298/02/NL/US and 15449/01/NL/Sfe. Jason Brazile acknowledges the support of Netcetera AG, Zurich.

References

i Nieke J., K.I. Itten, J.W. Kaiser, D. Schläpfer, J. Brazile, W. Debruyn, K. Meuleman, P. Kempeneers, A. Neukom, H. Feusi, P. Adolph, R. Moser, T. Schilliger, M. Quickelberghe, J. Alder, D. Mollet, L. Vos, P. Kohler, M. Meng, J. Piesbergen, P. Strobl, M.E. Schaepman, J. Gavira, G. Ulbrich, and R. Meynart. APEX: Current status of the airborne dispersive pushbroom imaging spectrometer. In: SPIE Earth Observing Systems IX, volume 5542, pages 109.116, 2004.

ii J. Brazile, P. Kohler, and S. Hefti. A software architecture for in-flight acquisition and offline scientic post-processing of large volume hyperspectral data. In Proceedings of the 10th Tcl/Tk Conference (Tcl/2003)*: July 29.Aug 2, 2003, Ann Arbor, Michigan, USA*, Dexter, MI, USA, 2003. Noumena.

iii D. Schläpfer, J. W. Kaiser, J. Brazile, M. E. Schaepman, and K. I. Itten. Calibration concept for potential optical aberrations of the APEX pushbroom imaging spectrometer. In SPIE Sensors, Systems, and Next Generation Satellites, volume 5234, pages 221.231, 2003.

iv J. W. Kaiser, D. Schläpfer, J. Brazile, P. Strobl, M.E. Schaepman, and K. I. Itten. Assimilation of heterogeneous calibration measurements for the APEX spectrometer. In SPIE Sensors, Systems, and Next Generation Satellites, volume 5234, pages 211.220, Barcelona, Spain, 2003.

v D. Schläpfer, J. W. Kaiser, J. Nieke, J. Brazile, and K. I. Itten. Modeling and correcting spatial non-uniformity of the APEX pushbroom imaging spectrometer. In 13th Annual JPL Airborne Earth Science Workshop, 2004.

vi D. Spinellis. Socketpipe. *http://www.spinellis.gr/sw/unix/socketpipe*, Visited May 2005.

vii Brazile J., D. Schläpfer, J. Kaiser, M. E. Schaepman, and K. I. Itten. Cluster versus grid for large-volume hyperspectral image preprocessing. In SPIE Atmospheric and Environmental Remote Sensing Data Processing and Utilization: an End-to-End System Perspective, volume 5548, pages 48.58, 2004.

viii Brazile J., Grid hack. *http://sourceforge.net/projects/ghack*, Visited May 2005.

ix T. Kientzle. libarchive. *http://www.freebsd.org/cgi/man.cgi?query=libarchive*, Visited May 2005.

x T. Kientzle. pkg add. http://www.freebsd.org/cgi/man.cgi?query=pkg add, Visited May 2005.