Cluster versus grid for large-volume hyperspectral image preprocessing

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

[a] Remote Sensing Laboratories, Dept. of Geography, University of Zurich,
Winterthurerstrasse 190, CH-8057 Zurich, Switzerland

[b] Centre for Geo-Information, Geo-Information and Remote Sensing, Wageningen University,
Droevendaalsesteeg 3, 6708 PB Wageningen, Netherlands

ABSTRACT

The handling of satellite or airborne earth observation data for scientific applications minimally requires pre-processing to convert raw digital numbers into scientific units. However depending on sensor characteristics and architecture, additional work may be needed to achieve spatial and/or spectral uniformity. Standard higher level processing also typically involves providing orthorectification and atmospheric correction. Fortunately some of the computational tasks required to perform radiometric and geometric calibration can be decomposed into highly independent subtasks making this processing highly parallelizable. Such "embarrassingly parallel" problems provide the luxury of being able to choose between cluster or grid based solutions to perform these functions. Perhaps the most convenient solutions are grid-based, since most research groups making these kinds of measurements are likely to have access to a LAN whose spare computing resources could be non-obtrusively employed in a grid. However, since many higher level scientific applications of earth observation data might be composed of more highly interdependent subtasks, the parallel computing resources allocated for these tasks might also be made available for low level pre-processing as well. We look at two modules developed for our prototype data calibration processor for APEX, an airborne imaging spectrometer, which have been implemented on both a cluster and a grid leading us to be able to make observations and comparisons of the two approaches.

Keywords: hyperspectral, cluster, grid, MODTRAN, APEX

1. INTRODUCTION

The Airborne Prism EXperiment (APEX) is currently being built as a demonstrator and calibrator for potential future European spaceborne imaging spectrometers.[1] In addition to supplying well-calibrated imaging spectroscopy data for use in earth observation applications, the APEX project also supports development of new processing algorithms and techniques.

One desirable technique to investigate is the possibility of the distribution of large volume computation to multiple compute nodes in a way that can be integrated into a standard product operational framework.

While processor speed and input/output capacity are growing at an impressive rate, processing algorithms are becoming increasingly more computationally intensive. A historical view of processing times for a typical airborne imaging spectrometer data acquisition is shown in Table 1. We believe the trend will continue in the diction of more complex computation in standard product generation. Therefore, we start to look at particular pieces in a standard production chain that can already easily themselves to a distributed computing solution.

Send correspondence to Jason.Brazile@geo.unizh.ch

Table 1. Downward trends in operating times vs. upward trends in processing complexity

Year	Download/Archive	Product Generation
1987[2]	9 working days	60 working days
1992[3]	5 working days	5-10 working days
1996[4]	3-4 working days	5-7 working days
1998	2 working days	

(a) Published AVIRIS "full flight tape" processing times

Module	Per Scene Per CPU
Parametric orthorectification	1-2 days
Atmospheric LUT generation	1-2 days
Feature based calibration	1-2 day
Spatial/Spectral uniformity	1 day

(b) Estimated time for selected APEX modules

2. PROCESSING MODULE CANDIDATES

A high level overview (up to level 2) of the data processing chain being developed for the APEX processing and archiving facility is shown in Figure 1.

Raw data files are provided on tape (or alternatively transferred from the acquisition computer via gigabit ethernet) and combined with data obtained from the calibration home base, a level 0 product is produced consisting primarily of segregated image cube scenes and laboratory-based calibration parameters.

Minimally, these level 0 products are used by the APEX PAF to produce radiometrically calibrated (but not spatially/spectrally uniform) level 1A at-sensor radiance. When spatially/spectrally uniform level 1 data is desired (at the expense of resolution lost due to re-sampling), uniform level 1C data can also be produced.

An optional, work-in-progress processing path involves automatic detection of anomalies such as smile/frown[5] and keystone[6] derived from the spectral measurements themselves rather than from laboratory calibration data. This requires the generation of a large lookup table (LUT) used for analysis of Fraunhofer lines and atmospheric absorption features[7] as an additional high resolution data calibration source.

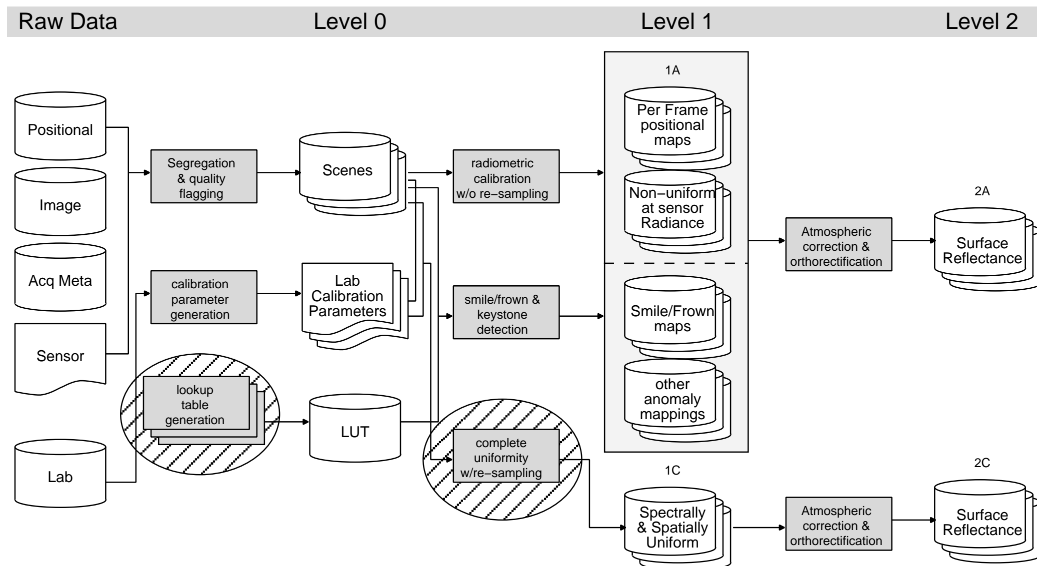


Figure 1. Simplified overview of the APEX airborne pushbroom imaging spectrometer data processing chain and 2 specific modules addressed in this work.

2.1. Atmospheric Lookup Table Generation

The use of a lookup table has long been used to ease the computational requirements of retrieving surface reflectance from airborne and spaceborne hyperspectral data. Common dimensions of such a table consist of attributes such as wavelength, pixel position, atmospheric water vapor content, aerosol optical depth, and terrain elevation.[8] A parametric fit of table entries can be used instead of running jobs on a per pixel basis and yet remain within respectable margins of error.

Since the work for generating this table is already done for level 2 atmospheric processing, it was determined that the same work can be used to provide an alternative high resolution source of calibration data based on known atmospheric features and Fraunhofer lines. One such use of this data is detection of spectral line curvature.[5] A typical lookup table generated for this use might involve generating entries that vary water vapor values over a range suitable for the given scene. One such table as generated by the ISDAS[9] image analysis system involves hundreds of calls to a radiative transfer program such as MODTRAN. In operation, this stage of processing typically requires 2-3 working days and was therefore chosen as a candidate for speedup via distributed computing.

2.2. Radiometric Processing

The current scheme for generating the APEX radiance data product involves radiometric data calibration as well as an optional step for assuring spatial and spectral uniformity. Since APEX provides programmable spectral binning, an initial step of un-binning must also be performed. This leads to the following processing sequence[10]:

- Undo binning
- Correct the readout smear effect of the VNIR detector
- Correct for dark current
- Invert pixel response to physical units
- Replace bad pixels
- Correct straylight and optionally, ghost images
- Optionally correct spectral and spatial non-uniformity
- Redo binning

The uniformity of pushbroom imaging spectrometers is affected by spectral and spatial misregistration within each detector array as well as by coregistration between the VNIR and SWIR spectral range. Since the correction of this non-uniformity involves interpolation/re-sampling, the resulting reduction in resolution suggests that correction may be undesirable in certain situations. Therefore two possibilities for APEX level 1 products have been defined as shown in 1.

In the 1C product where full uniformity is desired, preliminary analysis shows that uniformity processing is another suitable candidate for a distributed solution since the current code is decomposable into independent frame-by-frame subtasks.

3. PARALLELIZABILITY ANALYSIS

When analyzing a problem for discovery of potential parallelizability, there are a few standard characteristics that can measured which can give some idea of parallel computation potential. Two such characteristics are examined below.

```
36782 ktrace   RET   ktrace 0
36782 ktrace   CALL  execve(0xbfbffb0f,0xbfbffa18,0xbfbffa20)
36782 ktrace   NAMI  "../bin/modtran"
36782 modtran RET   execve 0
...
36782 modtran CALL  fstat(0x2,0xbfbff910)
36782 modtran RET   fstat 0
36782 modtran CALL  ioctl(0x2,TIOCGETA,0xbfbff8b4)
36782 modtran RET   ioctl 0
36782 modtran CALL  fstat(0,0xbfbff900)
36782 modtran RET   fstat 0
36782 modtran CALL  ioctl(0,TIOCGETA,0xbfbff8a4)
36782 modtran RET   ioctl 0
36782 modtran CALL  fstat(0x1,0xbfbff910)
36782 modtran RET   fstat 0
36782 modtran CALL  ioctl(0x1,TIOCGETA,0xbfbff8b4)
36782 modtran RET   ioctl 0
36782 modtran CALL  stat(0xbfbfd9d0,0xbfbfd920)
36782 modtran NAMI  "modroot.in"
36782 modtran RET   stat -1 errno 2 No such file or directory
36782 modtran CALL  stat(0xbfbfd9d0,0xbfbfd920)
36782 modtran NAMI  "MODROOT.IN"
36782 modtran RET   stat -1 errno 2 No such file or directory
36782 modtran CALL  stat(0xbfbfd980,0xbfbfd8d0)
36782 modtran NAMI  "tape5"
36782 modtran RET   stat 0
36782 modtran CALL  access(0xbfbfd980,0)
36782 modtran NAMI  "tape5"
36782 modtran RET   access 0
36782 modtran CALL  readlink(0x80e91b4,0xbfbfd8d0,0x3f)
...
36782 modtran CALL  open(0xbfbfd980,0x2,0x1b6)
36782 modtran NAMI  "tape5"
36782 modtran RET   open 3
36782 modtran CALL  fstat(0x3,0xbfbfd8e0)
36782 modtran RET   fstat 0
36782 modtran CALL  stat(0xbfbfd980,0xbfbfd8d0)
36782 modtran NAMI  "tape5"
36782 modtran RET   stat 0
36782 modtran CALL  fstat(0x3,0xbfbfd8d0)
```

(a) Output of system call trace for `modtran` process

```
...
enum iosyscalls {_OPEN, _CLOSE, _FSTAT, _LSEEK,
_ACCESS, _STAT, _READ, _WRITE};
...
#define _LSEEKSIZE 5
...
int ops[] = {
 _FSTAT,2,0x0,                    /* "stderr" */
 _FSTAT,0,0x0,                    /* "stdin" */
 _FSTAT,1,0x0,                    /* "stdout" */
 _STAT,(int) "modroot.in",2,
 _STAT,(int) "MODROOT.IN",2,
 _STAT,(int) "tape5",0x0,
 _ACCESS,(int) "tape5",0,0x0,
 _OPEN,(int) "tape5",0x2,0666,0x4,
 _FSTAT,4,0x0,                    /* "tape5" */
 _STAT,(int) "tape5",0x0,
 _FSTAT,4,0x0,                    /* "tape5" */
 _LSEEK,4,0x0,0,0x0,              /* "tape5" */
 _OPEN,(int) "tape6",0x2,0666,2,
...
};
int main(void)
{
    int i, size = sizeof(ops) / sizeof(int);
    for(i=0; i<size;) {
        switch (ops[i]) {
...
        case _LSEEK:
            if (lseek(ops[i+1], ops[i+2], ops[i+3]) < 0) {
                fprintf(stderr, "error: lseek 0x%X %s\n",
                        ops[i+2], strerror(errno));
                exit(1);
            }
            i += _LSEEKSIZE;
            break;
...
        }
    }
}
```

(b) Generated C code for I/O trace playback

Figure 2. Determining what amount of a process' total run time is due to I/O

3.1. Input/Output

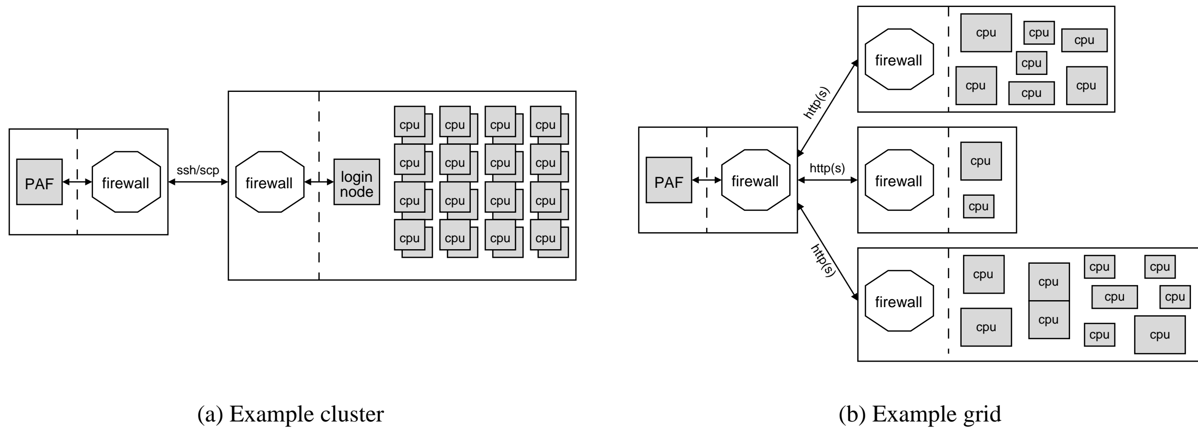
One of the most useful program characteristic to analyze is a program's runtime I/O profile. If a program spends the majority of its time reading and/or writing files then there is no benefit to splitting computation among multiple processors.

If I/O is found to be the bottleneck, standard techniques can be used to attempt to improve it such as buffer sizing/ data blocking, re-arranging calls such that access is mostly sequential, putting heavily used files on faster local media, etc.

Most modern operating systems provide the capability to trace system calls and produce a report as in Figure 2a. In some cases, the system call tracing facility can even provide profiling estimates on the amount of time spent in system calls. However, system call tracing allows the opportunity to measure I/O in a more direct way. It is possible to write a small program that automatically converts the output of a system call trace into a trace playback system that makes exactly the same calls with the same arguments in the same sequence. An example of such automatically generated C code used for playback is shown in Figure 2b. By subtracting the time taken to run the I/O trace playback from the total runtime of the unmodified program, it is possible to estimate an upper bound on the amount of potential speedup possible due to distributed computation.

The I/O tracing example shown in Figure 2 was used to determine that for our typical runs of MODTRAN jobs, I/O buffering could be easily improved, but is not necessary since only 5% of the program's runtime is spent in I/O.

Once time spent in I/O is reduced as much as possible, then analysis of computation can be addressed.



(a) Example cluster

(b) Example grid

Figure 3. Cluster vs. Grid model of parallel processing

3.2. Subtask Granularity

Once it has been determined that the majority of a process' runtime is spent on computation as opposed to I/O, it is then necessary to determine the possibilities for task decomposition. It might be possible to avoid the large amount of effort needed to analyze a programs' data processing structure if a problem's granularity is large enough.

For example in our case of running hundreds of MODTRAN jobs, it is clearly possible to view the problem at the high level granularity of indivisible MODTRAN jobs (or even collections of multiple MODTRAN jobs). In contrast, for some of the work performed for AVIRIS data processing,[11] smaller granularity was desired and MODTRAN itself was modified so that single instances could be run on a distributed cluster of up to 32 nodes.

4. CLUSTER

4.1. Overview

The cluster model of computing is a compromise between the more desirable view of a single global supercomputer and the lesser desirable view of clearly individual machines requiring each interaction to be explicitly designed and programmed. It arose from the need to work on large problems yet also to take advantage of the speed of computational advances occurring in the less specialized commodity computer market.

The cluster architecture used on the Matterhorn cluster[12] for our operational prototype is based on the ROCKS cluster distribution software[13] and is outlined in figure 3(a).

4.2. Development View

The standard programming model on most clusters is based on a Message Passing Interface (MPI) library. A single program is written that is executed on all nodes simultaneously. When interaction between nodes is required, those nodes call communications routines that cause synchronization barriers signaling all nodes to wait until every other node involved in that communication is ready to proceed. The cluster is used most efficiently if fewer synchronizations needs to be performed - in other words, if nodes can run their programs independently.

The communication architecture of a cluster can be setup such that high internode bandwidth is optimized, or low internode latency is optimized, or subgroup communication is optimized. Each of these models is the most appropriate for some set of computation tasks and perhaps less appropriate for others. Different hardware vendors provide optimized versions of the MPI library that is the most suitable for their communications hardware.

In any case, a cluster is usually available "as is" and the end-user has only limited options on the way a subset of nodes can be configured for a particular computation.

However, one major advantage is that due to the desire to make the most of an expensive shared resource, compilers and programming libraries available on cluster based systems are typically high quality.

4.2.1. Lookup Table Generation

In our lookup table generation module, the program flow is roughly outlined in 4(a). Since subtasks are highly independent, low internode communication latency is not needed, but high network bandwidth is helpful since 3MB of data is returned in the generation of each lookup table entry.

Another advantage of highly independent subtasks is that the number of nodes operating together toward the end result can be easily selected at runtime via a command line parameter. The difference between running a job on 8 nodes and running on 508 is decided by a command line parameter.

Because the task is linearly scalable up to 1700 nodes, we would want to run on the largest number of nodes available. However, the way the Matterhorn cluster's queuing system is set up, fewer node jobs are more likely to run sooner than several node jobs - even if the several node jobs would be able to run in a very short amount of time.

As shown in table 4(b), an 8-node job waited about 15 minutes in the queue in order to run a 20 minute job. The same job submitted to run on 64 nodes has been waiting on the queue for 14 days so far, even though the job would probably run in less than 5 minutes.

4.2.2. Radiometric Calibration

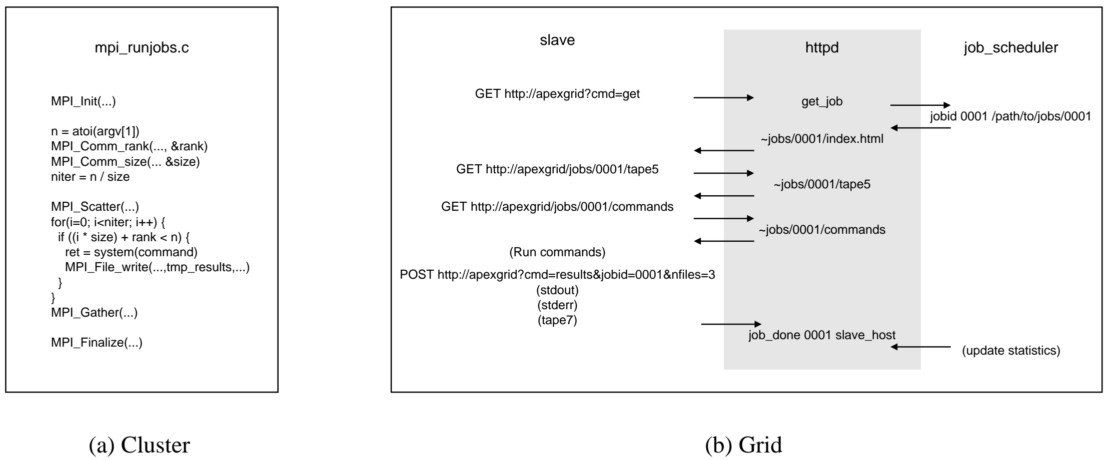
The distribution of computation for the APEX distributed radiometric level 1 processing module would be organized similarly, where job level granularity is on a frame-by-frame basis. However, an operational problem is that the current implementation is written in a high level data modeling language that requires a runtime license, of which none of which are available on the cluster. Even if some were available, it would likely be fewer than the number of nodes available in the cluster, leading to another scarce resource to be scheduled.

In some cases, some such modeling language environments have the ability to generate standalone binaries that run without the need for a license manager, but are operationally crippled using such techniques as requiring a mouse click in a splash screen even for non-graphical programs. Technical workarounds for such inconveniences are available (such as automated GUI event playback software) but the use of these may be in violation of software licensing agreements.

Other workarounds for such problems involve using open source "clones" of such modeling languages which may be compatible enough to run a particular application.

4.3. End-User View

An end-user performing an operational task expects a single centralized interface and reproducible processing times. The first of these expectations can be handled with appropriate integration and automation, assuming the remote cluster is accessible from the machine running the operational interface at least via ssh or HTTP.



(a) Cluster (b) Grid

Figure 4. Cluster vs. Grid program flow

However, in typical cluster environments, reproducible processing times are usually difficult to achieve since clusters are typically driven by a queuing system that attempts to use the system as efficiently as possible - which in practice means it is not immediately available for running a job at the time it was received.

4.4. Summary

The running time (as opposed to elapsed real time) of a cluster-based job is usually among the best achievable. Clusters typically have up-to-date computing nodes, good I/O subsystems, higher bandwidth/lower latency communication subsystems and better compilers and development libraries than are typically available otherwise.

If a job is relatively large and requires a lot of internode communication, running on a cluster might be the only feasible possibility. However, because of the competition for a shared resource in high demand, consistent, dependable access to the cluster cannot be relied upon.

5. GRID

5.1. Overview

For medium to large computing problems that don't require a high level of low-latency communication, workstations are powerful enough that the majority of them could be used simultaneously as personal workstations as well as employed in a lose computing grid. It is clearly a trend that as workstations and network infrastructure grow increasingly less expensive, more workstations will be available on a LAN than can be interactively used by personnel. The grid model of computation intends to take advantage of these unused resources.

A high level view of the grid used in our operational prototype is shown in the figure 3(b). It was composed of machines on a university departmental network, a medium-sized software development company, and a private home network.

5.2. Development View

There has been a great deal of recent interest in grid and peer-to-peer computing.[14] Unlike cluster computing there is no clear standard or even agreed upon standard model. Most research focuses on the generic case – conceivably a single global grid, where possibly smaller virtual grids automatically construct themselves, work on particular problems, and then disband making way for other problems.[15] This generic case needs to deal with a great many issues including resource discovery, network topology, data dissemination and collection, scheduling, security, error recovery, and accounting.[16]

Additionally there are a few user-driven grid frameworks that are gaining moment such as[17] which is being used by SETI@home and protein folding research groups, and the XGrid[18] product developed by Apple.

However, until the field becomes mature enough that a unique and ubiquitous generic solution becomes obvious for a majority of uses, there will continue to be many different possibilities for implementing grid based solutions based on the capabilities and possibilities presented by each situation.[19]

5.3. APEX Grid Framework

We implemented our own grid infrastructure using three small (each 200 lines of code) components as shown in figure 4(b).

First, we wanted to be able to communicate through firewalls since we want to use resources available in more than one institution. This results in typically only 2 choices of transport - either HTTP or ssh. We chose HTTP, since it is the most ubiquitous. If higher security is needed, HTTPS support could be added with minimal effort.

Therefore, the main grid component is a web CGI-based program that can connect via socket to an internal host running a job scheduler and feed these jobs to computation nodes via HTTP request on the client side.

Clients periodically poll the server for jobs, so it is a pull model of computation rather than a push model where servers initiate jobs on the clients.

The client component is a small program that emulates a web browser. It makes an HTTP request to a web server to request a job, fetches URLs corresponding to the input files, runs the computation, and then uses file upload via CGI POST to send back the results. When the web server gets the results, it forwards them to the job scheduler which maintains statistics.

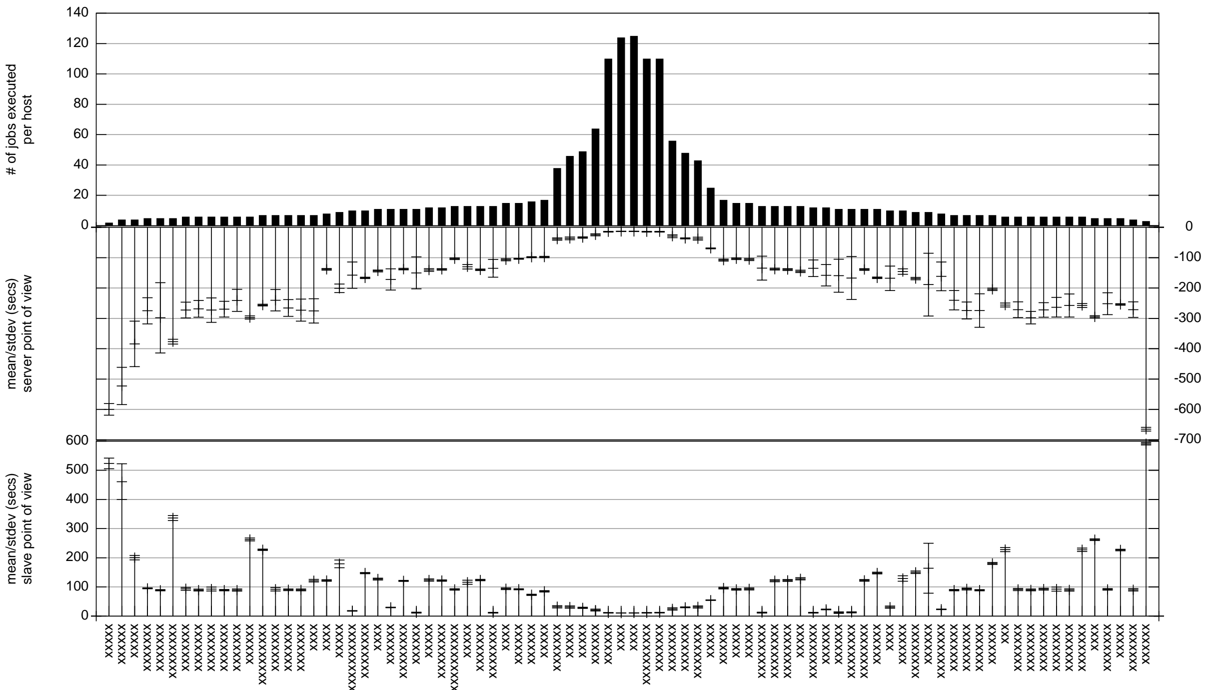


Figure 5. Statistics gathered on a 1700 job run on a grid of 82 heterogeneous nodes (host names elided)

The main reason for a separate job scheduler is to serialize access to the jobs. In addition to tracking statistics about which job is currently running on which client, it serves as an in memory cache of running/waiting queues. The job scheduler handles only single, short connections so no locking is needed. If a CGI request is not able to connect to the job scheduler, the web server requests the slave to sleep and try again later.

From the job developer's point of view, no specialized code is needed, but there are a small number of expectations of a job. A job is run in a subdirectory, called from a shell script with the reserved name ”commands”. The standard output and standard error from running these commands are written to files with the reserved names ”stdout” and ”stderr”.

5.3.1. Lookup Table Generation

In our lookup table generation module, a job maps directly to a single MODTRAN run. Since jobs are independent, no intranode communication is needed and therefore jobs run on a compute node one after another asynchronous to jobs running on other compute nodes (unlike in the cluster model). In a highly heterogeneous grid, this is vital. Since the fastest compute node in grid can run 40 jobs during the time it takes for our slowest node to run 1, it is important to not require the fast node to synchronize with slow nodes.

The runtime results of LUT generation on a grid of 82 nodes is shown in figure 5. The top third of the graph shows the distribution of jobs among compute nodes. The graph makes it clear that a small percentage of node perform the largest amount of work. As seen from table 4(c), adding more, slow compute nodes might not improve overall job time and could even slow things down by causing more network contention.

There are other interesting features of this graph. The bottom third shows mean and standard deviation of run times as measured on compute nodes, while the middle third of the graph shows the same information as measured on the server side. This shows that with this type of job, there can be a heavy penalty for being on a remote machine as compared to being on the local machine.

For this type of job, chances are that limitation to a few high performance nodes will give the most desirable results.

Table 2. Cluster vs. Grid Summary

user	+	Usually highest performance CPUs.
user	+	Also works with interdependent or low-latency communication requirements.
coder	+	Usually highest performance compilers and libraries available.
user	–	Large jobs may have unlimited waiting times in queue.
user	–	Runtime flexibility can be limited (e.g. license managers).
coder	–	Obtrusive application modification is usually necessary.

(a) Cluster characteristics

user	+	No upper bound on number of CPUs.
user	+	No special privileges needed to run.
coder	+	Sometimes possible to run applications unmodified.
user	–	Requires multiple copying of input/output files causing longer run times due to I/O.
user	–	Execution environment not as standardized as with clusters
coder	–	Wide variation in client resources and performance.

(b) Grid characteristics

5.3.2. Radiometric Calibration

Due to the runtime license limitation described above, our grid based distributed radiometric calibration model is feasible but limited. The number of running nodes can never exceed the number of licenses available which is usually measured in the 10s. During the day, licenses are needed for interactive workstation use, so nighttime operation is a further common limitation for operational use.

It is often argued that data modeling languages should be used for prototyping and operational code should be reimplemented in a more performant (and less license limiting) environment. If no new development is expected, this can be a good strategy. However, if any continuing development or maintenance is planned, it is usually best to keep the application closest to the language understandable by the domain experts as possible.

5.4. End-User View

As previously mentioned, an end-user performing an operational task expects a single centralized interface and reproducible processing times. Both of these are addressed with the APEX grid framework. Since an HTTP based transport is used, the control system and the job scheduler can be easily run from different geographic locations. Since jobs are not synchronous, large jobs can be run at the same time as small ones preventing job starvation commonly experienced in cluster environments.

5.5. Summary

While not as performant as cluster based systems, a grid-based system will probably provide more consistent run times and is theoretically more scalable, since adding more compute nodes only involves running the client on a new machine connected to the Internet.

6. CONCLUSION

Two example hyperspectral processing modules have been implemented in two different distributed computing environments - in a homogeneous Beowulf-style cluster and an ad-hoc heterogeneous grid. Each environment has advantages and disadvantages (summarized in table 2) but both are workable. Perhaps the most important result is that it does not require much effort to develop prototypes investigating both possibilities.

With minimal effort, we were able to demonstrate speed up of operational processing of look up table generation from about half a day to about half an hour (Table 3). Given the queuing system of the cluster that we had available, using 8 nodes seemed to provide the best efficiency due to longer queue waiting times for larger sub-clusters, even though theoretically 522 nodes were available. Coincidentally, a relatively low number of grid nodes (12) seems also to have provided the best efficiency due to relative speed of those nodes vs the much slower 70 remaining nodes used.

The remaining task of investigation include ensuring that distributed modules can be smoothly and robustly integrated into a complete operational process.

Table 3. Sampling of run times for lookup table generation based on 1700 MODTRAN jobs

# jobs	time (s)
12	10
892	11
605	12
170	13
7	14
9	15
1	17
1	18
1	23
1	28
1700	05h 28m 41s

(a) Single (fast) CPU

# CPUs	time	
8	26m 56s 42m 18s	(runtime) (total time)
64	? > 14 days	(runtime) (total time)

(b) 522 node cluster[12]

# CPUs	time
52	34m 30s
82	34m 59s

(c) APEX Grid

ACKNOWLEDGMENTS

This work was partially supported under ESA/ESTEC contracts 16298/02/NL/US and 15449/01/NL/Sfe. Fruitful cooperation and discussion with Robert Neville and Karl Staenz of the Canadian Centre of Remote Sensing about feature based calibration and look up table generation led to much of the motivation of this work. The support of the University of Zurich and Netcetera AG is also gratefully acknowledged.

REFERENCES

1. J. Nieke, 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. van Quickelberghe, J. Alder, D. Mollet, L. D. Vos, P. Kohler, M. Meng, J. Piesbergen, P. Strobl, M. E. Schaepman, J. Gavira, G. Ulbrich, and R. Meynart, "Status of the airborne dispersive pushbroom imaging spectrometer APEX (Airborne Prism Experiment)," in *SPIE Missions and Sensors*, W. Barnes and J. Butler, eds., **5542**, 2004. To appear.
2. J. Reimer, J. Heyada, S. Carpenter, W. Deich, and M. Lee, "Airborne visible/infrared imaging spectrometer AVIRIS ground data processing system," in *SPIE Imaging Spectroscopy II*, **834**, 1987.
3. E. Hansen, S. Larson, H. Novack, and R. Bennett, "AVIRIS ground data processing system," in *Third Annual JPL Airborne Geoscience Workshop*, **1**, 1992.
4. M. Aronsson, "A review of the new AVIRIS data processing system," in *1998 AVIRIS Workshop*, 1998.
5. R. A. Neville, L. Sun, and K. Staenz, "Detection of spectral line curvature in imaging spectrometer data," in *SPIE Algorithms and Technologies for Multispectral Hyperspectral and Ultrspectral Imagery IX*, S. Shen and P. Lewis, eds., **5093**, pp. 144–154, 2003.
6. R. A. Neville, L. Sun, and K. Staenz, "Detection of keystone in imaging spectrometer data," in *SPIE Algorithms and Technologies for Multispectral Hyperspectral and Ultrspectral Imagery X*, S. Shen and P. Lewis, eds., **5425**, 2004.
7. B.-C. Gao, M. J. Montes, and C. O. Davis, "Refinement of wavelength calibrations of hyperspectral imaging data using a spectrum-matching technique," *Remote Sensing of Environment* **90**(4), pp. 424–433, 2003.
8. K. Staenz and D. J. Williams, "Retrieval of surface reflectance from hyperspectral data using a look-up table approach," *Can. J. of R. S.* **Vol. 23, No. 4**, pp. pp 354–368, 1997.
9. K. Staenz, T. Szeredi, and J. Schwarz, "ISDAS - a system for processing/analyzing hyperspectral data," *Can. J. of R. S.* **Vol. 24, No. 2**, pp. pp 99–113, 1998.
10. 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*, JPL Publications, March 2004.
11. P. Wang, K. Y. Liu, T. Cwik, and R. Green, "MODTRAN on supercomputers and parallel computers," *Parallel Computing* **28**, pp. 53–64, 2002.

12. “University of Zurich Matterhorn cluster,” *http://www.matterhorn.unizh.ch* , Visited Jan 2004.
13. “Rocks cluster distribution,” *http://www.rockclusters.org* , Visited July 2004.
14. D. P. Anderson, “Public computing: Reconnecting people to science,” in *Conference on Shared Knowledge and the Web*, (Madrid, Spain), 2003.
15. I. Foster and C. Kesselman, “Globus: A metacomputing infrastructure toolkit,” *The International Journal of Supercomputer Applications and High Performance Computing* **11**, pp. 115–128, Summer 1997.
16. I. Foster, C. Kesselman, J. Nick, and S. Tuecke, “The physiology of the grid: An open grid services architecture for distributed systems integration,” 2002.
17. D. P. Anderson, “The berkeley open infrastructure for network computing,” Visited July 2004.
18. “Xgrid,” *http://www.apple.com/acg/xgrid* , Visited July 2004.
19. J. Ledlie, J. Shneidman, M. Seltzer, and J. Huth, “Scooped, again,” Feb 2003.