P2.22 DESIGN STUDIES FOR REAL-TIME DISTRIBUTED PROCESSING OF INFRARED HYPERSPECTRAL IMAGE DATA

Raymond K. Garcia* and Maciej J. Smuga-Otto

Space Science and Engineering Center, University of Wisconsin-Madison

Faced with the challenge of enormous data volumes specified by forthcoming imaging infrared interferometers, and the necessity of processing this data in a timely fashion, we have conducted design studies and built prototypes of a distributed data processing system capable of meeting throughput and latency requirements. Such a system must take in raw instrument data (interferograms of scene and reference black bodies) and produce radiometrically calibrated infrared spectra, as well as vertical atmospheric profiles retrieved from these spectra, including temperature and water vapor mixing ratio. The system must also take advantage of the inherently parallel profile of the data processing, and be designed to survive many generations of computing hardware evolution. We illustrate the current iteration of our test system, together with the problems encountered and the solutions devised. Applications to current ground and airborne interferometers as well as future spacecraft instruments are also presented.

1. INTRODUCTION

La plus ça change... An introductory meteorology textbook from 1970 mentions the problems associated with classifying and storing the enormous amounts of data (cloud photographs, as it happens) generated by the then-active Nimbus and Tiros weather satellite series [Barry and Chorley, 1970; p.60]. Today's satellites generate incomparably more high-quality data than those from 1970, and future satellite projects promise to continue this trend.

One such instrument project, the Geosynchronous Imaging Fourier Transform Spectrometer (GIFTS), is the focus of our study [Smith et al. 2002]. The GIFTS is designed to gather high-resolution infrared spectral data from a square grid of 128 x 128 sensor fields of view at a time, every such grid to be transmitted down from the satellite to the front-end processor on the ground once every 11 seconds at peak throughput. This translates to a continuous compressed data rate of about 60 Mbps (million bits per second), or well over a Terabyte of raw data a day. The University of Wisconsin Space Science and Engineering Center (UW SSEC) is performing design studies of the GIFTS data processing system for NOAA.

2. CONSTRAINTS AND CHALLENGES

The main challenge for designing a data processing system for GIFTS is to ensure that it can keep up with the realtime data inflow, while assuring correctness of the product. Since the GIFTS is a Michelson interferometer with an array detector, the calculations involved very closely resemble those needed for current atmospheric interferometer-sounders. These include correction for detector nonlinearity, removal of frequency scale dilation ('off-axis' correction needed as a result of optical path variation), radiometric calibration against reference blackbody spectra, and a finite field of view correction. Further design considerations mandate that damaged data must be corrected or discarded, and quality metrics must be extracted during the calculations.

Estimates of the computational cost of the needed science algorithms stand at approximately 200ms of compute time per instrument pixel at typical CPU and memory bus speeds for calendar year 2001. That computational cost is predicted using theoretical cycle count calculations and verified empirically using profiles of existing instrument data processing. In order to process such data in an acceptable amount of time, it becomes necessary to maintain the option of using large-scale parallel computing systems.

Significant cost savings can be achieved for large-scale computing projects by replacing special purpose supercomputers with clusters of commodity off-the shelf (COTS) processors where possible. The combined processor power of such 'Beowulf clusters' can mitigate their comparatively poor processor interconnection capability. The highly independent nature of the calculations required by each detector pixel fits squarely in the capability envelope of clustered systems. Therefore, the data processing system for GIFTS (henceforth called the GIFTS Information Processing System, or GIPS) is targeted to be deployed on such a cluster of commodity processors, necessitating careful planning to ensure that the performance requirement is met, and giving rise to other requirements specific to a distributed computing environment.

Chief among these requirements is robustness—a system deployed across a network of commodity machines, albeit tightly coupled, must be tolerant to the inevitable failure of some of the commodity components. Some faults in the system can be handled on the hardware side by the system integrator—by carefully monitoring common health indicators like processor and disk drive temperature—but the lion's share of the

* *Corresponding author address:* Ray Garcia, 1225 W. Dayton St., University of Wisconsin-Madison, Madison, WI 53706; rayg@ssec.wisc.edu

responsibility rests on the shoulders of the software system engineers.

Other requirements for such a system stem from a need for longevity, easy maintainability and operability. These requirements reduce the cost of running the system. Given that GIFTS is intended as a research platform for a future generation of satellite-borne high spectral resolution interferometer-sounders, the concepts, designs and source code used in GIPS must be comprehensible to programmers not associated with the initial deployment.

The requirements for the GIFTS Information Processing System present an imposing challenge that will need to be overcome in order to make full use of future generations of meteorological satellites.

3. SCIENCE INTRODUCTION—ALGORITHMS, PIPELINES IN THE ABSTRACT

GIFTS' heritage includes instruments and associated science processing algorithms such as UW SSEC's uplooking Atmospheric Emited Radiance Interferometer (AERI), NASA LaRC's NPOESS Airborne Science Test-bed Interferometer (NAST-I), and UW SSEC's airborne Scanning High-resolution Interferometer Sounder (S-HIS). The GIPS system interface must specify a consistent set of software Application Programming Interfaces (API's) with which to couple these science algorithms to the data to be processed by the system. This level of abstraction is necessary reduce design complexity, and to facilitate future upgrades to the science algorithms in a deployed system.

Concurrent with designing the interface is the task of adapting the existing science algorithms to that interface specification, and testing the algorithms separately (unit tests), as an ensemble (functional tests), and in the target environment (integration tests). Since these algorithms form the core of GIPS, and since managing them presents an inherent source of complexity, a unifying metaphor is desirable to help grasp the salient points of the architecture.

Since most of the processing is straightforward in the sense that there is little or no feedback based on the computations already performed, the chosen metaphor is that of a "refinery." The data can then be thought of as being successively refined by a series of stages in a numerical pipeline—these stages being the individual science algorithms which act upon the data.

This metaphor encourages a functional programming approach to the task of data processing: Each *numerical operator*, representing a science algorithm, is a stateful entity accepting a particular set of immutable data structures as input and providing another particular set as its output. Thus, numerical operators can be combined together into complex *stages* which transform one kind of data into another by the application of several compatible operators in a defined order. Such modularity goes a long way to satisfying maintainability and documentation requirements by allowing the inclusion of an audit trail service. This audit service records which operations were performed on a specific packet of data.

3.1 Science—blackbody and calibration example

An example of a set of refinery components is the linear calibration stage. The purpose of this numerical operator assembly is to assign physically referenced radiances to the instrument measured intensity values. Each spectrum is an array of *(wavenumber,value)* pairs prior to the conversion, and is an array of *(wavenumber,radiance)* pairs upon exit from this stage.

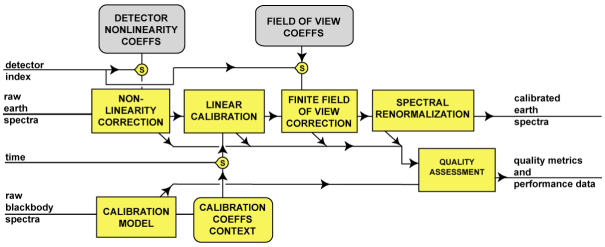

Figure 1. Flow diagram of simplified radiometric calibration stage.

The conversion is effected by mapping the measurement intensity at each wavenumber to a physical radiance, using simultaneous blackbody reference spectra of known temperature as a measurement intensity scale. A limitation of the instrument design, however, is that the blackbody reference spectra cannot be taken simultaneously with the measurement itself, and must be estimated.

The blackbody reference model can be represented as a fit curve over time for each wavenumber. The source data for building the reference model are obtained by periodic measurements of reference blackbody sources through the instrument optics. On aircraft and ground-based instruments, these curves are often temporally interpolative: this is largely due to the instability of the instrument optics being significant in comparison to the desired measurement accuracy. For GIFTS, a predictive model for maintaining the fit curves is appropriate and necessary in order to meet system performance and capability requirements.

Given this, the operating pipeline for a given instrument pixel includes periodic updating of the calibration model by supplying blackbody spectra of known temperature for that pixel. The calibration model operators in turn provides a database of fit curve coefficients which are applied to a measurement time offset in order to synthesize reference spectra. The intensities of the reference spectra are used in the calibration equation to assign physical radiances to the measurement intensities recorded by the detector pixel. [see Figure 1]

The implication of applying the coefficients is that the calibration stage must maintain state information (the fit curve coefficient 'context'), and that logic must be capable of deciding whether to update the coefficients, or use them to create radiances, based on engineering information from the instrument.

The use of predictive fit curves invites later verification of the accuracy of the prediction for use as a quality metric, for model tuning, to provide radiance post-correction, or to test for a need for re-computing of the radiance spectra at that measurement time. These considerations mandate a system requirement to allow scheduled reprocessing.

4. SYSTEM REQUIREMENTS AND DESIGN: WHERE SCIENCE AND DATA MEET

The task of applying the science algorithms to the relevant data in the context of the entire processing system brings with it new considerations and challenges, including testing for correctness, control mechanisms, and most important, the ability to meet performance requirements.

The next challenge lies in running 16384 (128 x 128) such pipelines with sufficient concurrency to ensure that the processing system is keeping up with the data inflow rate and meeting downstream needs for low product latency (i.e. the time between reception and delivery).

The current specification of the GIFTS instrument asserts that the processing of one pixel of GIFTS data is numerically independent of the processing of another pixel—that is, there is no need to communicate intermediate data between parts of the system working on spatially distinct pixels. Pixel-specific reference data structures such as resampling intervals for off-axis correction, nonlinearity settings for the pixel, and current blackbody-derived calibrations to apply during the science processing need to be available to the pixel pipeline, but this sort of data coupling is significantly easier to manage than a spatially-coupled system would be.

Even with this simplifying assumption for which communication would only need to be established between a 'master' scheduling and data distribution point and 'workers' assigned to push pixel data through the pipeline, the volume of generated network traffic for both data and control is formidable. To ensure that the system meets requirements, the performance characteristics are broken down into two fundamental metrics:

Throughput is defined as the rate of data flow through the system. Throughput provides a measure of the system's capacity to keep up with real-time data inflow. In order to ensure sufficient throughput, the interconnection layer between CPUs must be able to sustain the expected data rates. If this network layer can sustain the load, then system throughput can theoretically be increased by adding more computing hardware to an existing configuration, with the caveat that certain parts of the software may execute longer as the configuration size grows. This so-called *super-serial* term (serial component performance penalty) may be introduced by resource limitations in the master scheduling software, by system-wide synchronizations, by interconnection delays and error recovery, by certain classes of contention-resolution protocols, or by complete network graph startup and shutdown procedures in a communicator-based system such as MPI. The effect is that the system's performance can be expected to peak at a certain node count after which it will decline as the serial component penalty outweighs the contribution of any additional computing resources. Using techniques that display such super-linear behavior should thus be limited to rare special circumstances such as fault recovery procedures, amortized over the processing of many data cubes, or avoided altogether.

Latency is the other crucial system performance metric used to ensure that data products are delivered in a timely fashion after initial receipt of raw instrument data from the satellite. Whereas throughput requirements can simply be met by adding more nodes to the cluster and ensuring that pathological scaling behaviors don't offset the performance gain thus obtained, the latency requirements demand that the tasks to be processed be broken down into sufficiently small pieces, so that no single task stays "in the pipeline" past the maximum allotted time.

An example of how throughput and latency differ can be obtained from looking at two possible strategies for processing GIFTS data cubes—each data cube represents the total data collected from the 128 x 128 grid of individual detectors. At one extreme, data from each pixel is fed to a worker node of the cluster whenever it is freed up from processing a previous pixel. Using this "fine granularity" approach, latency of processing an entire GIFTS cube is limited only by system's total processing power. At the other extreme, an entire GIFTS cube's worth of data is sent to a worker to process at once—this is the coarse granularity extreme—when the next cube arrives on the system queue 11 seconds later, it is assigned to another (now free) worker. With a sufficient quantity of workers, this scheme ensures sufficient throughput to keep up with real-time data inflow. But the latency for each cube is then bound by the time it takes an individual node (in this case, a computer in the cluster) to process the entire cube. For modern hardware, that latency is on the order of 1 hour.

The latter approach is an unacceptable risk due to current technological limitations; the former, while outwardly appealing, can incur significant per-pixel transport overhead, and thus may contribute to the aforementioned super-linear performance ceiling. Thus this *granularity* of data messages becomes a tunable

parameter of the system, one that should be manipulated to account for a variety of implementation-specific effects and balanced to trade product latency for overhead cost.

4.1 System requirements and design—special concern: Robustness

A well-known property of parallel processing systems is that since the probability of failure of any one of the identical compute nodes is constant, the overall probability of compute node failure increases linearly with the number of nodes in the system. Thus, it becomes a priority to transform likely component failures from being system failures in to recoverable system faults.

Any system design having constrained latency (i.e. “real-time”) and availability requirements must include discovery of likely failure scenarios. These scenarios can be grouped into those involving bad hardware, bad software or bad data that is based on device specifications and past community experience with similar clusters. Anticipated failure scenarios can be used for formulation of mitigation strategies.

Mitigation strategies affect the choice of third-party hardware and software components as well as their intended use. The economic trade-off is between system programmer time needed to develop autonomic responses and thus reduce the component failure to a system fault versus operator time (both immediate and downstream) needed to intervene and recover from a failure versus the price to purchase and maintain specially hardened components not subject to the same sorts of failure.

Failures of hardware due to normal wear-and-tear can be expected of a commodity computing cluster. While interconnect failures are rare, recovery from such a failure is potentially time-consuming. These kinds of problems may necessitate hardware redundancy within the system and the ability for the software to identify faults and compensate using redundant hardware without increasing latency. Alternately, some failure paths, such as a power failure or interconnect failure, may be relegated to operator intervention combined with an acceptable recovery time.

A wide spectrum of mitigation strategies can be identified for a software system having a requirement for effectively uninterrupted service. At one extreme, a component failure would cause the software to terminate gracefully, to reconfigure appropriately, and to restart at an earlier checkpoint known to be good. The other extreme involves building a system that is capable of adding and removing components dynamically, and using this capability to reassign tasks ad hoc. The former approach runs the risk of bringing the amount of system downtime past explicitly set limits; the latter necessitates a more complex system and increases costs, especially if performance is to be maintained during the various dynamic switchovers. Other strategies can include fail-over, keeping a spare on hand for rapid install, or even re-routing the entire raw data stream off-site through contingency bandwidth to a secondary system. Such a standby system could interrupt its normal duties to temporarily handle the data processing while the primary system is brought back on-line.

A compromise might allow dynamic failover to be implemented for those components, such as pixel computation nodes which are most likely to fail often, while a less choreographed technique might be utilized for less frequent faults, such as a database server failure which in any case may necessitate human intervention.

The mitigation strategy for system faults must include a means of notifying the system operator and recording faults for the purpose of localization and backtracking. The originating component must also be identified for purposes of detaching, servicing and reattaching, but trying to establish the root cause of component failure may be an unjustifiable expense (e.g. for older equipment), or best left to long-term statistical methods. It will be beneficial for the system to maintain meaningful records of its activities for use in analysis, reconstruction and verification.

4.2 System requirements and design—special concern: Audit trails

The algorithms, and the ways in which they are combined, will evolve over the operational lifetime of the system. In order to distinguish data processed by one rather than another variant of an algorithm, an audit service is needed to enrich the product pipeline outputs with metadata. At its simplest, this service will record what software was applied to the data, and in what order. Of particular importance will be the version signatures of the algorithm code and contributing data structures, as well as identifying the participating hardware. This allows for tracking down potentially dubious data should a flaw in a particular version of an algorithm or reference data element be discovered. Should a problem be identified in a hardware component, only the directly affected data would have to be marked as defective. This data could then be scheduled for re-processing using the system's reserve capacity.

5. SYSTEM DIAGRAM AND OVERVIEW

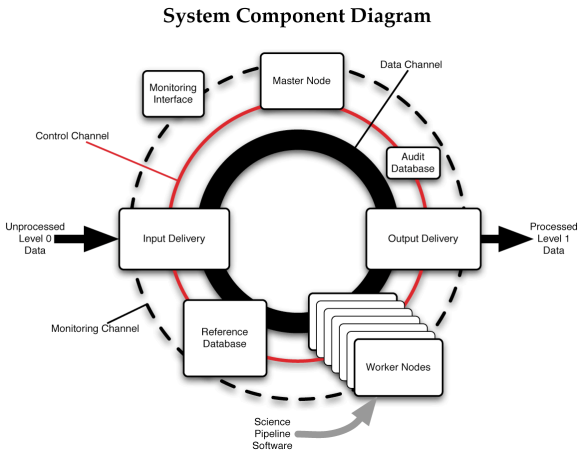

Figure 2. Abstract component diagram for GIPS system candidates.

Figure 2 lays out the abstract components and connections of a family of tractable distributed computing system designs intended to address the design issues raised in the previous sections. The data channel between nodes, for example, would need to be wide (i.e. carry large volumes of data), fast, and have low latency. The control channel would need to be fast enough to match the data channel and highly reliable. The monitoring channel would not need to be as fast or as timely, but should be easily separable and not interfere with either the data or the control channel. The input, output, audit, control and worker services could be fairly independent of each other as long as they communicate reliably on these channels.

The breakdown of the overall problem into these discrete components helps in the study of available technologies for each piece and for the selection of an optimal solution for each component in isolation. The distributed computing environment for data delivery could be Java/RMI or C++/MPI while the control and monitoring channels could be written in Java/RMI, C++/ACE, or even a high-level scripting language such as Python using IP datagrams. Likewise, the audit subsystem may be implemented in one of the already listed technologies or as an independent SQL server to be used by the other components, or as a set of XML log files on a storage area network.

6. TRADE STUDY OF AVAILABLE COMPONENTS - GOING TO MARKET WITH THE SHOPPING LIST

One of the most fundamental choices, which would drive much of the rest of the system design, is choosing a distributed computing platform. There are many candidates: Java was built almost from the ground up to be a network programming language, but much of the emphasis of its native networking library (RMI) is on ease of use and robustness with less focus on the resulting performance. Another heavyweight contender is CORBA which, like RMI, tries for the illusion of locally accessible but remotely served objects. It does so at the cost of simplicity. Another object-oriented framework is C++/ACE (the Adaptive Computing Environment) which wraps up network communication patterns in simple API calls, but is only beginning to be adapted on Beowulf clusters for the kind of problem being addressed here. Another approach is to use message-passing C/C++ libraries like MPI, which provide good performance on a variety of interconnects at the cost of runtime malleability. Finally, it is possible to build a custom solution using network primitives such as sockets, RPC (remote procedure calls), datagram packets, and native hardware libraries for specialized interconnects such as Myrinet or Infiniband.

Once the platform for the main data and control channels is established, other smaller scale design studies still need to be performed. There is always a need for support data to be maintained across the system, such as for transferring the states of pipeline operators for checkpoint, diagnostic or migration purposes. The choice of how to do this ranges from employing another ORB system such as RMI/CORBA, to an SQL database such as PostgreSQL, or to implementing it as a web service. The system operator user interface is another area with great component design variability. The operator console could be written in any of Java/Swing, C/GTK, C++/MFC, or as browser-based web application.

7. RESULTS TO DATE

7.1 First iteration studies

A first iteration of the development spiral resulted in test code designed to explore the practical aspects of ensuring sufficient throughput and latency. It relied on simplified operator interfaces, and testing concentrated on ways of monitoring the performance of both the network layer and underlying communication API's for purposes of tuning and evaluation.

The underlying platform for this experiment was the MPI messaging library used from within a C++ codebase on a Linux/x86 cluster of homogeneous nodes with gigabit ethernet interconnect. Instead of trying to build a complete self-sustaining 24/7 solution, a subset of the problem was chosen that involved the explicit feeding of one or more GIFTS datacube-shaped data sets to a processing pipeline.

7.2 Approach

In order to run this simulation early in the design process, two fundamental steps needed to be taken. First, a realistic profile of the execution footprint (both processor occupancy and wall-clock time) of a hyperspectral instrument processing pipeline was obtained. The existing Scanning-HIS processing code

provided the required science pipeline, and common UNIX utilities including Gprof were used to obtain such a profile. Although the eventual science pipeline deployed for the GIFTS may differ from this implementation, it is doubtful that it would be less efficient, and thus invalidate the results of this study.

Once the profile was obtained, placeholder operators were constructed which took up equivalent amounts of processor cycles, or equivalent delay of clock-time, as the science algorithms. Part of the reason for this was to dissociate the study of performance from much more exhaustive tests eventually needed to assure correctness. These computing cost-equivalent black boxes can then be inserted into a test framework and interfaces can be refined during development without necessitating a change in the science libraries.
Another crucial reason for creating these time-equivalent black boxes is to allow the construction of a large virtual cluster on a much smaller physical cluster. Thus many copies of the pixel-worker code which would reside on separate nodes in the target system may co-exist on the simulation cluster if they are running science module blackboxes which just sleep for a period instead of taking up CPU time. The result should also represent a “worst-case” behavior, since the virtual workers are potentially vying for (and thus serializing) overhead cycles on the available physical hardware.

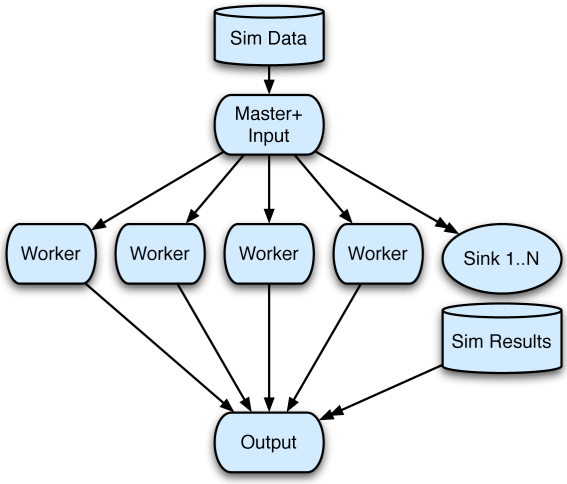

Figure 3. Flow diagram for 'Osaka' performance demonstration.

This was the architecture chosen of the first iteration code experiment, dubbed ‘Osaka’. It consisted of an Input/Master node, an output node, and a configurable number of worker nodes which could either be made to occupy an entire physical node (in which case they ran a CPU-occupancy equivalent) or which could be grouped many-to-one physical node - a "sink" (in which case they ran the wall-clock time equivalent) [See Figure 3].

A fundamental consideration in designing the algorithm harness for the worker nodes was to ensure that the communication framework tasks could run in parallel with the science computations, minimizing the number of worker cycles wasted waiting for work to be assigned. This was achieved by using MPI's nonblocking messaging modes [see Figure 4].

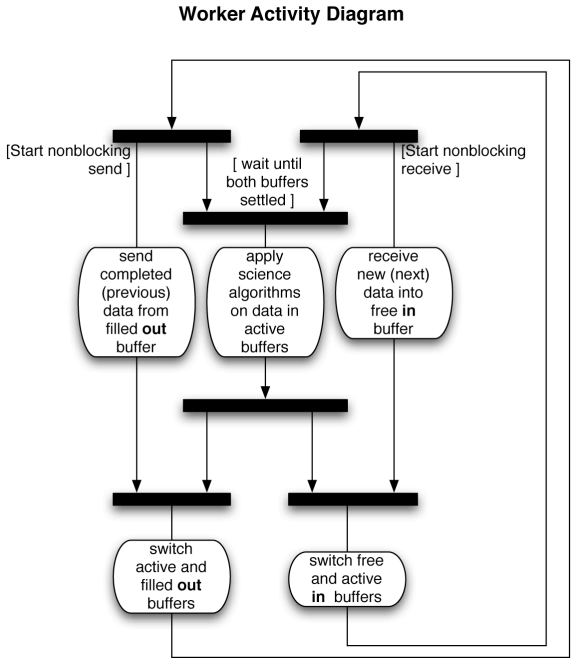

Figure 4. UML activity diagram for workers in Osaka, showing asynchronous send, receive combined with computations.

7.3 Observations

Various configurations of the Osaka test framework were run on two different implementations of MPI—MPICH and LAM. MPICH was accompanied by a profiling library, MPE. This allowed the gathering of performance measurements of the relative amount of time each node spent on various kinds of tasks. The MPE profiles were subsequently visualized using the Jumpshot application [See Figure 5]. In order to corroborate the behavior observed on the Jumpshot diagrams and to allow for simple monitoring of LAM runs, a low-overhead manual profiling library was also included in the code. The data thus obtained provided the basis for the performance evaluations for this initial development cycle.

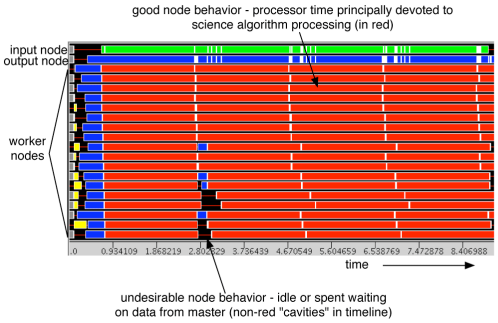

Figure 5. Annotated Jumpshot performance profile from Osaka running a small number of virtual worker nodes.

Observations from Osaka indicate that the startup/shutdown costs were directly proportional to the number of virtual workers engaged in the run – in accordance with the super-serial model of performance. Thus adding nodes to the configuration did not result in a proportional increase in performance but seemed to reach an asymptote around 160 nodes, above which which the system would become unstable. Since this was above the number predicted necessary, it was not a source of immediate concern. Starting up a job with more data cubes queued up would amortize this startup/shutdown cost, and would push the system towards a more linear scaling behavior [See Figures 6 and 7]. In addition, it was observed that LAM provided more predictable behavior when faced with simulated faults, including the forced shutdown of worker nodes during a run by killing the associated process.

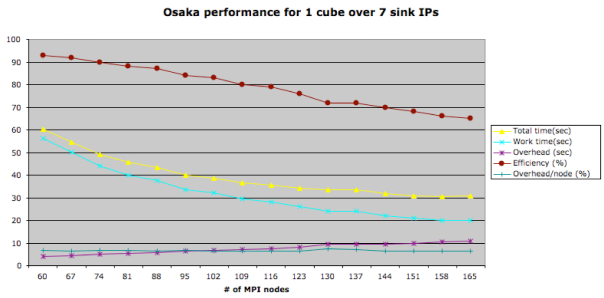

Figure 6. Efficiency measurement as a function of node count running a single data cube through Osaka. Note degradation of efficiency due to linear increase in start-up cost.

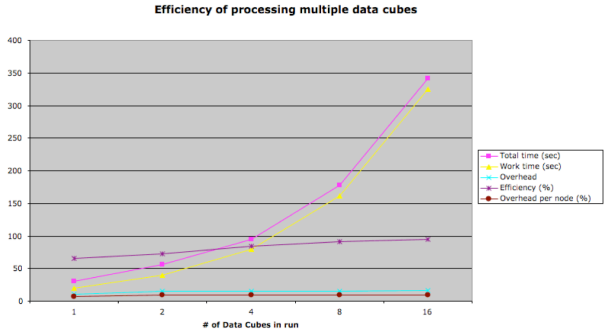

Figure 7. Efficiency of Osaka as a function of number of data cubes processed. Mitigation of super-linear cost is accomplished by amortizing it over multiple data cubes.

7.4 Second iteration studies

The second iteration design studies (in progress as of writing) include investigation of the ACE network computing framework for the control and data channels with a particular eye on meeting robustness objectives simultaneously with performance objectives. This includes, for instance, graceful admittance and dismissal of worker resources.

Further, the specification and integration of one or more simplified algorithm stages doing real work on simulated GIFTS data cubes into this test code is in progress. This will include straw-man API specifications for a reference database and algorithm operator interfaces and test specifications derived from heritage code and draft theoretical basis documents.

8. CONCLUSIONS

The GIFTS instrument's data processing requirements pose a significant challenge to the atmospheric science and computer science fields. The University of Wisconsin - Madison Space Science & Engineering Center, with significant support from NOAA and its contractors, is aggressively working to contain the complexity of the task in an incremental design and implementation strategy. This strategy includes research into integrating proven off-the-shelf hardware and software technologies in order to of provide a scalable and reliable distributed data processing system capable of delivering timely products to the meteorology community. By unfolding the dependencies and algorithms of the science software into an array of sixteen thousand data processing pipelines hosted on a commodity cluster of workstations within a task distribution framework, we demonstrate that the challenge can be made computationally and organizationally tractable.

9. ACKNOWLEDGMENTS

The authors acknowledge helpful discussions with Keith McKenzie and Anand Swaroop of NOAA as well as Shahram Tehranian and Tony Harvey of AC

Technologies, Inc. Support for this work was provided by NOAA federal grant NAO7EC0676 with previous support under NASA contract NAS1-00072.

10. REFERENCES

Barry, R. G. and R. J. Chorley, 1970: *Atmosphere, Weather and Climate. Holt*, Reinhart and Winston.

Gunther, Neil J., 1998: *The Practical Performance Analyst*. McGraw-Hill.

IEEE/EIA 12207 (1998): *Software life cycle processes*. Institute of Electrical and Electronics Engineers.

Minnett, P.J.; Knuteson, R.O.; Best, F.A.; Osborne, B.J.; Hanafin, J.A., and Brown, O: *The Marine-Atmospheric Emitted Radiance Interferometer: A high-accuracy, seagoing infrared spectroradiometer.* Journal of Atmospheric and Oceanic Technology 18 (6), 2001, pp994-1013.

Nittel, Silvia, Kenneth W. Ng and Richard R. Muntz, *Conquest: CONcurrent QUEries over Space and Time.* Nittel, Silvia with Kenneth W. Ng and Richard R. Muntz, UCLA.

Smith, W.L.; Harrison, F.W.; Hinton, D.E.; Revercomb, H.E.; Bingham, G.E.; Petersen, R., and Dodge, J.C. *GIFTS-The precursor geostationary satellite component of the future earth observing system*. In: IGARSS 02: International Geoscience and Remote Sensing Symposium, 2002. Proceedings, v.1. Piscataway, NJ, Institute of Electrical and Electronic Engineers, Inc. (IEEE), 2002, pp357-361.

Tobin, Dave; Baggett, Kevin; Garcia, Ray; Woolf, Hal; Huang, Allen; Knuteson, Bob; Mecikalski, John; Olson, Erik; Osborne, Brian; Posselt, Derek, and Revercomb, Hank. *Simulation of GIFTS data cubes*. In: Conference on Satellite Meteorology and Oceanography, 11th, Madison, WI, 15-18 October 2001. Boston, American Meteorological Society, 2001. Pp563-565.

Twomey, S., 1977: *Introduction to the Mathematics of Inversion in Remote Sensing and Indirect Measurements.* Elsevier Scientific Publishing Company.

Wallanu, Kurt C., Scott A. Hissam and Robert C. Seacord, 2002: *Building Systems from Commercial Components.* Addison-Wesley.