Scalability Analysis of the Multi-Look Time Domain Processor on XSEDE Compute Resources

Gregor von Laszewski, Fugang Wang, Geoffrey C. Fox
School of Computing and Informatics, Indiana University
Informatics West, 901 E. 10th Street
Bloomington, IN 47408
laszewski@gmail.com, kevinwangfg@gmail.com, gcfexchange@gmail.com

Jilu Li, John Paden
Department of Electrical Engineering & Computer Science, University of Kansas
1520 West 15th Street, 2001 Eaton Hall,
Lawrence, KS 66045-7608
jiluli@ku.edu

ABSTRACT

Without doubt the analysis of data from Polar Regions is an important aspect of identifying environmental impact by humans. The calculation of automatic techniques for determining ice and snow layer boundaries in radar echograms remains a hard problem because of two reasons. First, the data volume and the associated calculations to retrieve results are large requiring considerable supercomputing time. Second, the data itself provides challenges based on the high degree of noise, the often faint layer boundaries, and confusing linear structures caused by signal reflections and clutter. Thus it is necessary to optimize existing and future algorithms while improving the performance, but also introduce new techniques that combine together weak image cues, reasoning explicitly about uncertainty in both the evidence and the resulting layer boundary estimates.

In this paper we report on two important findings. First the performance improvement of the Multi-look time processor program, and second the introduction of a new analysis technique improving our image analysis.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]. G.1.0 [**General**] Numerical Algorithm, Parallel Algorithm.

General Terms

Algorithms, Measurement, Performance, Design.

Keywords

Polar Data Analysis, Ice Sheets, Sensor Data Analysis, XSEDE, Matlab.

1. Introduction

The Center for Remote Sensing of Ice Sheets (CReSIS) is a Science and Technology Center established by the National Science Foundation (NSF) in 2005, with the mission of developing new technologies and computer models to measure and predict the response of sea level change to the mass balance of ice sheets in Greenland and Antarctica. As part of this effort, the NSF's Science and Technology Center (STC) program combines the efforts of scientists and engineers to respond to problems of global significance, supporting the intense, sustained, collaborative work that is required to achieve progress in these areas. CReSIS provides students and faculty with opportunities to pursue exciting research in a variety of disciplines; to collaborate with world-class scientists and engineers in the US and abroad; and to make meaningful contributions to the ongoing, urgent work of addressing the impact of climate change [2]. As part of this effort we have recently focused on two activities:

- The performance optimization of the multi-look time processor
- The algorithmic improvement of our work published in [3].

The paper is structured as follows. In Section 2 we report on the performance optimization of the multi-look time processor. In Section 3 we report on the algorithmic improvement of our image analysis algorithm while applying them to ice-sheets and snow layers. The paper ends with our conclusions.

2. Seasonal Data Collection

The Center for Remote Sensing of Ice Sheets (CReSIS) is currently collecting up to 100 terabytes of raw data per field season using its radar suite over Greenland and Antarctica; new radar developments over the next few years will grow this number by an order of magnitude. After collection, the raw data is processed to produce data products such as flight line coverage maps, synthetic aperture radar (SAR) images, snow and ice layering information, and ice bed elevation maps. These data products are distributed to international cryospheric communities for use in a variety of climatic-related studies such as the estimation and prediction of the contribution of ice sheets to global sea level. CReSIS develops their own processing tools to process raw radar data and generate and publish the final data products for cryospheric communities. High-end computing resources and scalable cyber infrastructure are needed in order to process the large volume of data fast using complex algorithms.

SAR processors are a core component in CReSIS processing toolbox. The synthetic aperture radar (SAR) processor focuses the raw radar data to achieve fine along-track resolution. Two SAR methods have been implemented. The first is f-k migration (FK) method in frequency-wavenumber domain uses Fourier or fast-convolution techniques to accelerate the processing. FK requires that the raw data are uniformly sampled and that the platform travels in a straight line parallel to the ice surface. Since a straight line trajectory is never achieved in practice, motion compensation techniques are applied which modify the data so that it approximates a straight line trajectory. The ice surface must also be flat and is approximated by taking the mean surface height. The second known as multilook time domain processor (MLTDP) integrates subaperture processing into time domain methods. MLTDB split a longer aperture into many small apertures and the time domain SAR processor generate an image for each subaperture. The multiple looks of the same scene are finally combined to get an image which is equivalent to being focused by a longer aperture. The second method does not impose constraints on the flight path and ice surface and thus is more accurate leading to a better signal to noise ratio or improved focusing as shown in Figure 1. However computation of this method is several orders higher.

The first SAR processor is currently more often used by CReSIS because of it fast computation. The purpose of this proposal is to implement and run the MLTDP processor on high-end computing resources and scalable cyber infrastructure to deliver better data products to cryospheric communities.

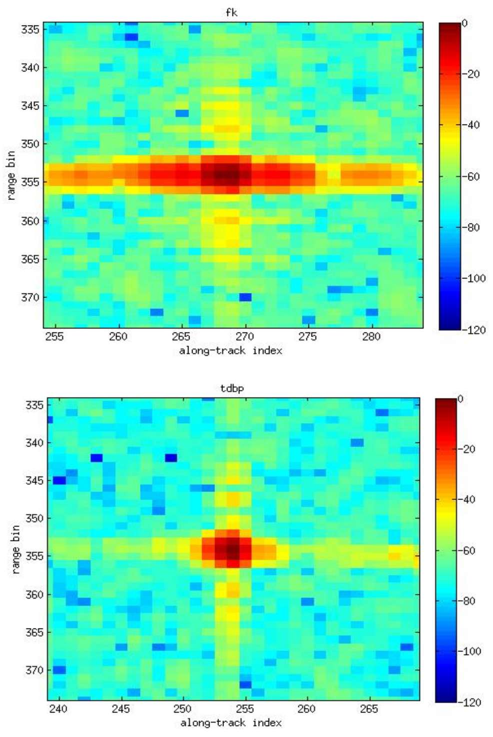

Figure 1: Point target focused by FK (left) and by MLTD (right) Processing.

3. Scientific Workflow Framework

The original algorithms used in this work are part of the CReSIS processing toolbox. This toolbox is written in matlab and provides the advantage that it leverages a programming tool and framework that is well established in the community, namely matlab. New algorithms can be developed by the engineeres and researchers and tested out as part of analyzing the data that is gathered during the seasonal data collection conducted in Antarctica and Greenland. However, utilizing matlab although convenient for the scientists and community brings forward the challenge that it may not perform as well as algorithms written in C or other programming languages. Furthermore, the workflow of the analysis is driven by a framework that interfaces with a queuing system of a supercomputer while hiding this aspect from the user. The input is controlled by a set of parameters that are manipulated in an Excel spreadsheet.

While analyzing this sophisticated scientific workflow, motivated by the scientific work to be conducted, we identified that through isolation of the most performance intensive parts of the workflow we can significantly speed up the analysis of the data while optimizing this portion. Furthermore, we identified that the tight integration of the queuing systems may prevent the utilization of a supercomputing resources in a computational Grid. We are addressing both issues and have made significant progress in this regard.

3.1 Towards a Mesh of Supercomputers, Grids, and Clouds with Cloudmesh

As we have extensive experience with Supercomputers, Grids and Clouds, we designed a framework that can utilize all of them to provide sufficient compute resources for our analysis. An important part of this is Cloudmesh that was developed to bridge between these environments and enables us to set up a virtual compute testbed as a service by dynamically integrating such resources. More details about cloudmesh are available in [4, 5]. However here we will focus on a feature of the cloudmesh design that can support the workflow of the CReSIS project. Instead of using the Karajan Workflow Engine of the Java Commodity Grid Kit we identified that it is for this project far easier to use the cloudmeh HPC interface. Our design allows us to manage thousands of supercomputing jobs on a set of heterogeneous supercomputers. The important differentiation here is that these computers do not have to be in the same security context to form a virtual organization managed by a Grid provider. Instead, the user can define its own Grid while adding resources from a variety of Grids. This is not a new concept and was introduced by the precursor to the Java CoG Kit. However, new is that we are now able to not only utilize resources form established computational Grids such as XSEDE but also from cloud environments. The cloudmesh framework allows for example to set up virtual clusters on OpenStack, AWS, Azure, HP Helion, and many other cloud providers enhancing the ability to run such compute intensive jobs also in cloud environments. We envision that on such virtual clusters we also can deploy on-demand job management frameworks such as OpenPBS or Slurm to simplify the distribution of the computational tasks. Through cloudmesh these tasks are than registered in a NoSQL database and their execution can be monitored. This will also allow for fault tolerance, as we can for our project predict how long it will take for a task to be completed on a particular machine. If a task is not returning in time it will be scheduled for re-execution. Such performance data can be identified during runtime and leverages our work reported in [6, 7]. One of the new efforts we are currently working on is to expand upon our design and work towards the integration of a REST service model that will make the integration of cloudmesh into matlab (or other frameworks and services) much easier. As cloudmesh is written in python we can therefore use our REST interfaces and enhance the CReSIS matlab toolbox while providing access to heterogeneous resources spawning Supercomputers, Grids, Network of Workstations, and Clouds. We are currently focusing on resources provided at XSEDE, Indiana University, University of Kansas, but want to leverage resources in Azure if given the opportunity and once we have demonstrated feasibility of the calculation engine in XSEDE and FutureSystems. The later already provides a combination of traditional supercomputing resources but also cloud resources. It will be soon enhanced by a cloud with more than 128 modern servers and 3456 cores. Each server has 128 GB of memory and an overall 0.5 TB of SSDs. This machine has been carefully designed to provide a smaller testbed that mimics the system offered by SDSC's comet machine. While leveraging this rich set of resources through a self managed virtual organization via the cloudmesh software we could offer the service also through the cloudmesh GUI and specialized matlab interface as a gateway to the project users. Thus it is clear that what we have created is not a traditional computational Grid but a *Mesh* that integrated these resources. Hence this motivated us to use the term *cloudmesh* for our software tool.

4. Performance Evaluation and Improvement of the CReSIS Multi-look Time Domain Processor

As discussed previously the CReSIS analysis system includes a multi-look time domain processor (MTTDP) [8]. We identified the portion of the algorithm that is used frequently but takes a long time to complete. Hence our optimization efforts are focused on the optimization of this part. The original algorithm was written in matlab. The code was analyzed for several possible code improvements and the improvements were implemented. These improvements include:

1) *Better alignment of data types in memory.* We found that several data types used in the algorithm did not correspond to the actual data types intended to be used such as complex numbers. Although this does not pose an issue within the original matlab code It does provide a challenge during the conversion of such programs to C. Thus we identified such data types and replaced them with datatypes that allow a seamless conversion to C. We also verified that the numerical calculations while performing these changes were not affecting the original or the modified program.
2) *Avoiding repeated calculations through variable substitution.* We found a sizable number of small calculations that were repeated in several parts of the algorithms. However as the code is executed tens of thousands of times, overtime this will result in an unnecessary time penalty. While replacing such calculations with variables and placing the calculations in appropriate loops we avoided a significant number of such redundant calculations while using our variable store.
3) *Replacement for key matlab functions with a C version of the code.* Once we have made these optimizations we took the resulting program and translated it to C. This translation undoubtfully leads to the biggest performance improvement, as we will see later. The translation was only possible after we carefully corrected the usage of the data types (see 1).
4) *Easy integration of the newly developed code.* To support the existing extensive CReSIS toolbox, the new optimized functions must easily be integratable in the code. This is achieved while wrapping the C function into a matlab MEX code that allows the calling of the C function without changing the original matlab code (other than the name of the function). Hence the CReSIS workflow is unchanged.
5) *Update of the Matlab ecosystem*. We also found that the current system was executed while using a fairly old version of matlab. We updated to a very recent version, which provided features that were not available in earlier versions. While using this version we were able to more easily generate a C program. While purchasing a variety of matlab coder licenses for a variety of systems we were also able to more easily generate makefile that could be adapted for a variety of systems.

The modified code was tested extensively for numerical accuracy and an independent verification was conducted at University of Kansas to double check the accuracy of the new code. Furthermore, we developed a code performance testing framework that used two data sets and tested accuracy with a variety of parameters. This framework allowed us to run the code in uniform fashion on various machines including supercomputers at University of Kansa, IU, and XSEDE.

The result was that on a modern hardware architecture the code was significantly faster than before and for a standard dataset with a bin size of 1700 the speedup between the original un-optimized original Matlab code running on Quarry (main production analysis machine) and new code was over a factor of 60 while comparing the performance of the original algorithm on quarry with the performance of the fastest server running the C optimized version. The compilation time was also decreased significantly by a factor of over 100 on a modern computer.

While using this new optimized program we conducted a number of significant performance studies on various architectures to identify the single core performance characteristics of the code. This is important as to identify how the code behaves and to outline a path forward in the production workflow management. The new code runs a factor of 9-20 faster on most machines while comparing the C vs the original matlab performance. Quarry, echo, and india only receive a performance improvement of about 5. The comparison of optimized code run times normalized to fastest machine is shown in Figure 3. Quarry, Karst (IU) and Drebber (KU) are used to analyze CReSIS data while Bigred, Bravo, India, Delta and Echo are servers at IU. Gordon is an XSEDE machine at SDSC and Optiplex and laptop are client testbeds.

5. Development Environment

Originally, we tried to conduct our work on the optimization of the algorithm on the supercomputers accessible to us. However we found that on the main machine Quarry (at that time still located at IU) the algorithm was unacceptable slow (motivating this work). Although other machines were faster we still faced the issue that for the optimization the execution time was still to long in order to make progress quickly. We looked at the algorithm and identified that much of the calculations are memory bound and would behave similar to an ODE. Therefore we could try to estimate on which machine we should conduct our optimization while consulting the build in matlab benchmarks. Hence we obtained a quick overview while executing the standard matlab benchmark on the machines we had access to at that time. The results of this benchmark are shown in Figure 2. The machines used in this study are in more detail described in Table 2. Interesting to note is that we immediately identified that the use of Gordon in no flash mode will be unsuitable. However with the use of flash it was very reasonable. However, this motivated us to also explore the lead authors Laptop which is a MacBook Pro (Mid 2014, a relatively recent machine. It is configured with 16GB flash main memory and has 8 cores. We purchased additional

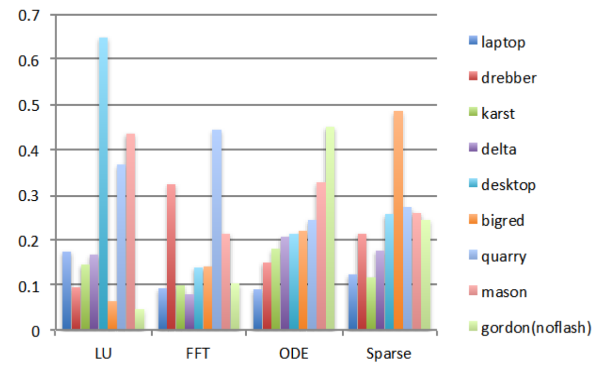

Figure 2: Matlab standard benchmark

licenses for this machine and other desktop machines. This allowed us to more easily use the graphical user interface features of matlab in the code development and debugging phase. Not surprisingly, the laptop performed the best from all machines we had access to. Hence, we used this machine to do all the development work needed for the code optimization.

To further simplify the development we also identified a smaller dataset and reduced the workflow to only execute a small number of samples, as well as reducing the number of iterations. Thus instead of waiting for hours while testing a single modification we reduced the time to less than one minute for our test data. We verified later that our original assumption about the capabilities of the machines derived from the standard matlab benchmark were giving us an accurate picture of on which it may perform well.

One of the other issues we had was that the licensing management of matlab deployed within IU. This was based on the issue that we were in the need of purchasing of matlab licenses that were not part of the typical licenses sold to other IU staff and especially the lack of access to the matlab documentation while using the IU group licensing. Without this access we could not access relevant documentation that required a login onto the Matlab portal. While consulting with Mathworks and the responsible IU staff this issue was eventually overcome. However, we believe this is a great lesson to learn especially in cases where XSEDE looks for alternative licensing models. One must be aware the documentation is limited to those registering in the Mathworks portal, but such a user must have a valid product license associated with the appropriate toolbox. We also realize based on our experience with the C compilation that version differences between various matlab versions may be significant. Hence, it may be necessary to support multiple matlab versions on hosted systems. Although users may bring their own licenses, as recently discussed with us in the resource provider XSEDE meetings, it may not be a solution for those that run production jobs due to cost. Naturally, compiled matlab code may be needed. Therefore, it is in the best interest of such sites to provide the tools *and* access to all documentation that makes it easier to optimize matlab code and translate it to C for anyone running matlab jobs. This not only includes the matlab compiler, but also must provide to the relevant users matlab coder license. In the long run the offering of these toolboxes will pay of as we can provide already see from our own experience in the optimization of our code discussed in this paper. However we also recognize that the development of such code is not trivial, but may require the interaction with computer scientists and experts familiar not only with matlab but with general software architecture. This will allow the scientists to focus on developing quality code in matlab while the experts can optimize it for the target machines. Furthermore it is not sufficient to just optimize the algorithm but the computer scientist must be able to evaluate the workflow and the interaction within it in order to identify possible future venues of software and resource utilization such as the once we have identified with cloudmesh. Thus the ideal support person must be multifaceted and provides significant expertise in software architecture to leverage possible other frameworks.

6. Performance Study

We have conducted a number of significant performance studies on various compute resources to identify the single core performance characteristics of the code. This is important as to identify how the code behaves and to outline a path forward in the production workflow management. It will enable us to answer the following questions:

- **Question A:** *How good is the currently used hardware and software for the code?*
- **Question B:** *How much faster is the code on a single modern core vs. the original code on for example quarry, the machine we are currently using*?
- **Question C:** *How much faster is a C code version of the code on available supercomputers?*
-

6.1 Performance Testing Framework

To do a meaningful performance study we developed a performance testing framework that includes an isolated version of the most time intensive calculation of the workflow. We verified that when running the entire worklflow the rest of the algorithm did not contribute significantly to the overall execution time. The framework allows us to utilize different test data and also to adjust for the *bin size* a parameter that influences the runtime significantly. This test framework was ported and deployed to all machines we had access to.

6.2 Hardware and C Optimization Impact

First, we need to identify how the existing original code compares to an optimized version. This will help assessing potential for optimizations of the code as well as identify how to proceed. For this test we ran on a single core the original code on the machine quarry. Quarry used to be located at the IU datacenter and used as one of the primary machines to run the analysis. Quarry is currently being moved to University of Kansa Obviously it is an older generation machine. We compared the performance of the computational intensive in matlab on quarry and the C optimized code on an OSX MacBook Pro (Mid 2014, 2.8 GHz Intel Core i7, 16 GB 1600 MHz DDR3, $3200). This laptop was the newest hardware we had access to. This comparison is important as to show how the current code performs vs the performance of a modern architecture while also considering code improvements. As there are many parameters and datasets to choose from we have taken one dataset and set the bin size to 1700 that was identified to be a scientific relevant parameter set for many calculations.

The results were astonishing, as we ***achieved a speedup of ~63 between the unoptimized original matlab code running on quarry and the Laptop on a single core.*** Hence we see that while using more modern hardware plus our optimization a significant performance improvement can be achieved that will drastically reduce the analysis time.

Naturally this comparison needs to be further analyzed an in our next comparison we identified that on the Laptop the performance improvement factor between the original code and the C code is ~23. We than compared the C code vs. the original code on quarry and only obtain an improvement of about ~5.1. ***When comparing the performance of the C codes between the Laptop and quarry we see that the Laptop runs 6.61 times faster.***

We conclude from this data that the algorithm is dominated by calculation speed and access to memory. Both are significantly better with newer architectures. Furthermore, we ran the newest matlab version on the Laptop.

6.3 Resource Comparison

To compare other resources we have run the same performance study on Quarry, Bravo, Echo, Delta, India, Echo, Bigred-2, Karst, Gordon, Drebber, and the Mac OSX Pro. The details of these machines are provided in Table 2.

We ran the following versions of the code, where resource indicates the name of the resource:

- **<resource>-p:** is the unmodified matlab production code executed on the resource
- **<resource-o>:** is the optimized matlab production code executed on the resource as explained in Section 4 2)
- **<resource-c>:** is the c code derived from the optimized matlab code executed on the resource.

Figure 3 shows the performance results on the various resources in seconds. The abscissa varies the bin size that is used as one of the main parameters of interest to the scientists. A useful value is dependent on the actual data and may need to be varied. We have currently identified a maximum of 1700 for this value and data set. This is a realistic value that will be used as part of our main analysis. In addition we provide in Figure 4 the speedup comparison of the fastest resource against all other resources and algorithms compared. Hence the MacPro is the baseline with one. All other machines perform slower than this resource by a factor specified by the ordinate. In the Figures we also provide performance data while varying the the bin size. We can make the following observations from the two figures:

1. While varying the bin size the algorithm behaves linearly.
2. In cases where divergence is found from the linear scalability of the calculation we find, that resource limitations on the server took place.
3. We can clearly identify the three groups of algorithms based on if they were (p) original (o) optimized (c) translated from the optimized matab code to C.

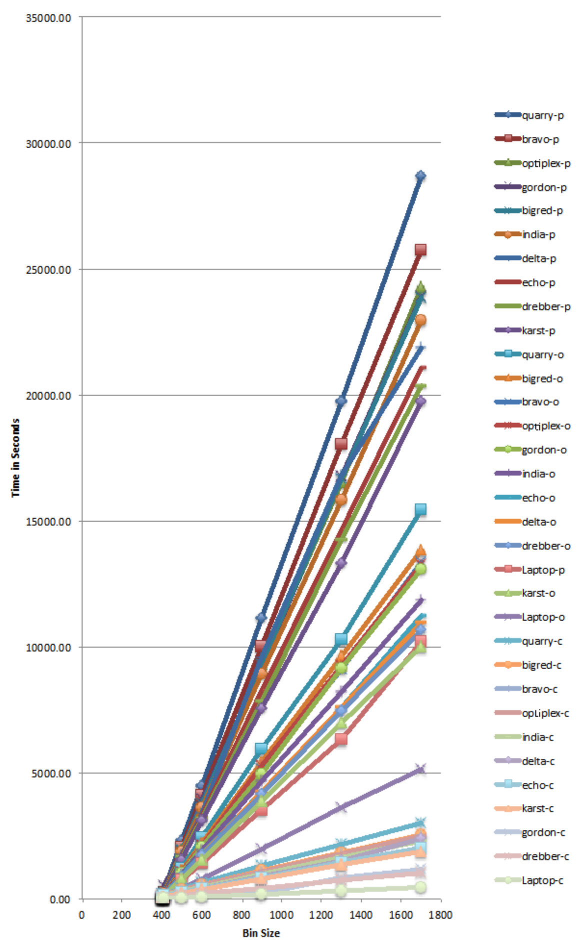

Figure 3: Runtime of the test data and performance measurement framework for the computational intensive part in the multi-look time domain processor.

To further analyze the data we present in the subsequent Figures selected details from Figure 3 and Figure 4.

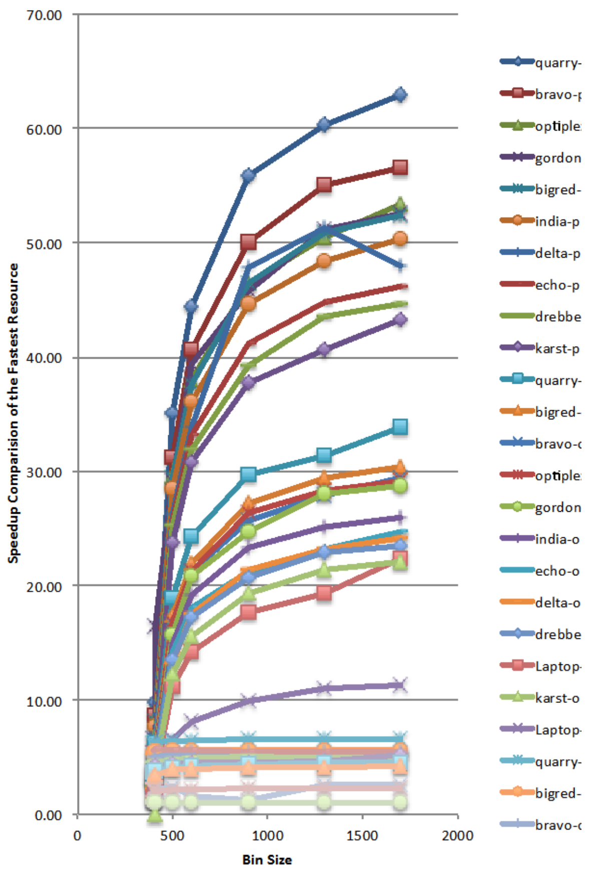

Figure 4: Speedup of the test data and performance measurement framework for the computational intensive part in the multi-look time domain processor.

6.4 Fastest Single Core Performance Analysis

The fastest single core performance is obtained on the MacBook Pro. This is not surprising as this machine is the newest machine, and has a relative powerful processor. Furthermore it has 16 GB of main memory and 1TB flash SSD. The details for the performance and speedup are depicted in Figure 5 and Figure 6. We identified that on this machine we can achieve an overall performance improvement is of a factor of over 20 while comparing the original optimized version of the algorithm with the optimized and to C translated version. The fast processor and access to fast memory are here of great importance.

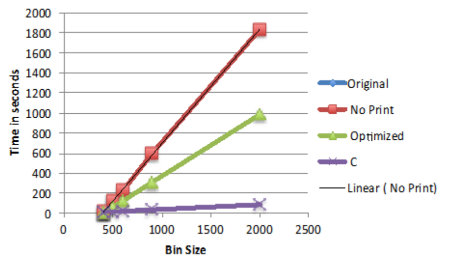

Figure 5: Performance Improvement of the original Multi-Look Time Domain Processor on the fastest single core resource

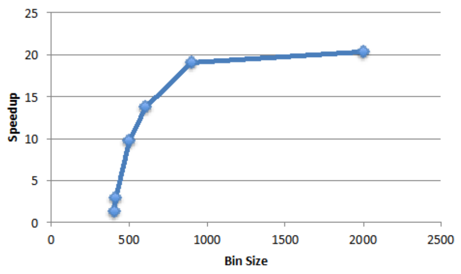

Figure 6: Speedup of the original Multi-Look Time Domain Processor while comparing the original algorithm with the C based optimized algorithm on the fastest single core resource.

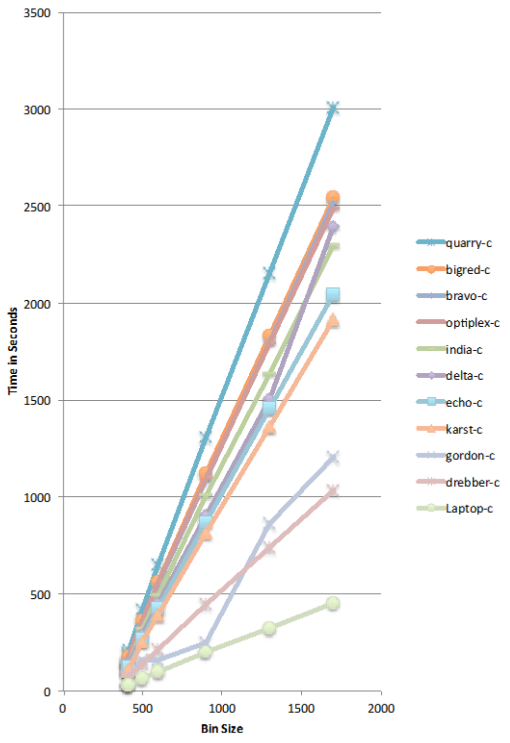

Figure 7: Runtime of the test data and performance measurement framework for the computational intensive part in the multi-look time domain processor while only comparing the optimized algorithm that was translated to C.

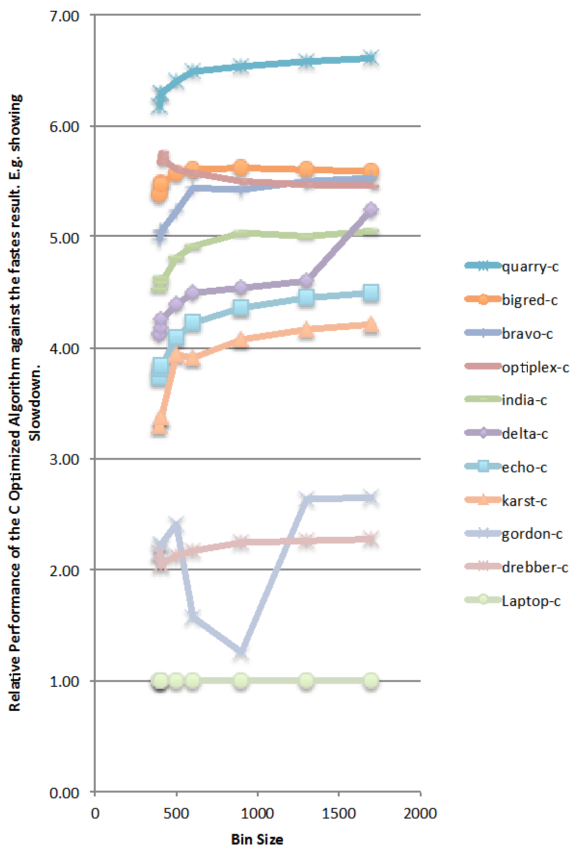

Figure 8: Speedup comparison of the optimized C versions on various machines.

6.5 Comparison of the Optimized Algorithm Translated to C

Figures 7 and 8 show the comparison of the optimized matlab algorithms translated to C. We see that for a bin size of 1700 that we are interested in the fastest single core performance next to the MacBook Pro is achieved by Drebber and by Gordon. We identified that the laptop is 2.27 times faster than Drebber and 2.65 faster than Gordon. All other machines are at least four times slower making a considerable performance impact. The worst machine of them is not surprisingly Quarry, the oldest of the resources, which is 6.61 times slower. However due to the availability of this machine it will be continued to be used to obtain production results. While observing the results in more details we find several noteworthy issues with some of the resources. First, on an optiplex 960 (an old desktop) we ran into an actual slowdown of the algorithm due to a lack of memory on the machine. Second, we see a significant slowdown of the algorithm with increased bin size on Delta. Also here the memory became problematic. Thus such machines are performing best with smaller bin sizes. Such bin sizes actually do exists in the workflow and therefore if we can identify them such machines could be restricted in the bin sizes that ought to be calculated on them. Third, we have to recognize that very small bin sizes may not be of interest and may not be relevant for the calculation. Thus we discard bin sizes smaller than 500. This will than transform the data for Gordon from a dip to a slowdown with larger bin sizes. Hence we can make the same statement that we have done: for Gordon as we are running in to some resource limitation with larger bin sizes. However as this machine is significantly faster than all the others, this is one of our primary target machines. At this time we have not yet undertaken an analysis on XSEDE's Stempede and other new resources that are not yet available to us such as comet.

6.6 Predictability of the runtime for Production Runs

Based on the data we hove obtained we can use it to fairly accurately predict the overall runtime of the workflow that is needed as part of the data analysis part. We have used the almost linear scaling even within obtaining the scalability data itself while we were predicting by running 3 small samples up front to identify when the larger example is completed. We used for this a regression analysis and thus given a resource and a bin size we can predict the runtime of this job. This data can now be used to predict the overall runtime..

7. Prediction of the Overall Data Analysis

The previous analysis focused on the optimization of the most computational intensive portion of the workflow to be conducted. It provides us with a very good basis to identify a very accurate prediction for the computation time on resources.

The field data collected by the team is measured by field seasons. A year has four field seasons. During a field season data is collected by plane. There are 20 flights in a field season. During one flight data is collected between 30-60 frames. A frame is defined over 50km with 2 waveforms and 15 channels. Furthermore a job is defined by 500m (this translates into a bin size between 400-1700, in our algorithm this is defined by the parameter 1700), one waveform and one channel. One frame will result in 3000 jobs of that are based on our optimized algorithm. Hence we calculate that in one field season we obtain between 600 to 1200 frames and between 1.8M to 3.6M jobs. While using our data from the previous section we determined the lower and upper bound of the core hours on a given compute resource. We see that on Gordon we will need approximately 1.2 Million core hours to calculate the field season.

In addition we will need 300K hours to calculate regions that we determine of special interest. It will also allow us to further optimize the algorithm while taking into account core, memory and other resource specific parameters. The additional core hours will also be used for other algorithm improvements that are discussed in the next section. Thus the allocation will not only serve as a production, but also as a development allocation. We also plan to conduct a performance analysis on Stempede and would like to work together with TACC staff to further explore the Matlab-Bring-Your-Own-License model. For that we currently anticipate 50K core hours for the multi look method and an additional 50K for the method discussed in the next section.

Table 1: Predicted core hours for one field season

	Karst	Drebber	Quarry	Gordon	Bigred2
1 job (hrs)	0.53	0.29	0.84	0.34	0.71
core hours - lower estimate	955,000	516,500	1,504,500	603,500	1,271,500
- upper estimate	1,910,000	1,033,000	3,009,000	1,207,000	2,543,000

In addition to Gordon's compute time we will also leverage Quarry. However, resource limitations especially in regards to memory may hinder progress. As the machine is currently not operational we were unable to make further recommendations. One thought would be to upgrade the memory or to consolidate the existing the memory into a smaller number of servers.

8. Feature Detection of Ice-Sheet Boundaries

Furthermore, we are investigating automatic techniques for determining ice and snow layer boundaries in radar echograms. While several recent papers have studied this problem including our own in [1], it remains a hard problem because of the high degree of noise, the often faint layer boundaries, and confusing linear structures caused by signal reflections and clutter. We thus need new techniques which combine together weak image cues, reasoning explicitly about uncertainty in both the evidence and the resulting layer boundary estimates.

We proposed a new technique for layer finding that removes many of the assumptions and restrictions of [1], while preserving the ability to integrate weak information and explicitly model uncertainty. This new approach was published in the IEEE Conference on Image Processing (ICIP 2014)[3]. In particular, the paper introduces several important contributions to improve both the accuracy and utility of layer-finding. Our technical innovation uses Gibbs sampling for performing inference instead of the dynamic programming (Hidden Markov Model)-based solver of [1]. This allows us to remove some of the assumptions of the probabilistic mode and solve for layer boundaries simultaneously, yielding automatic layer detection results that are significantly better than the approach in that paper. Unlike [3], which was really only practical for solving the two-layer problem (i.e. finding just the ice surface and bedrock layers), this new approach can handle an arbitrary number of layers, which will be important for finding internal (annual) layers in ice and snow (and we continue to investigate that problem). Moreover, the Gibbs sampler produces explicit confidence intervals, thus giving bands of uncertainty in the layer boundary locations. Since noise and ambiguity in radar echograms are inevitable, we believe that estimating confidence could be crucial in applications of layer identification (e.g. when used as input to glaciological models), and to our knowledge this is the first paper that has demonstrated this capability.

We tested our layer-finding approach using a 826 echograms from the 2009 NASA Operation Ice Bridge program, which was the same dataset used by [1] so we can directly compare our accuracy. Figure 1 shows results on three sample echograms, presenting the output of our technique (including the confidence interval) as well as the ground truth and baseline technique of [1]. Quantitatively, compared to human-labeled ground truth, our technique outperforms [1] significantly, by decreasing the error rate (measured in terms of mean or median squared deviation from ground truth) by about 44.3% for surface boundaries and 48.3% for bedrock. Our technique is slower than [1] (about 17 seconds per image compared to a few tenths of a second), but since layer finding is trivially parallelizable across images, we believe accuracy is much more important than compute time in practice. We also quantified how informative the confidence intervals are by computing the percentage of ground truth layer points that are contained within the estimated intervals. We found that 94.7% of the surface boundaries and 78.1% of the bedrock boundaries are within the intervals, for an overall percentage of 86.4%.

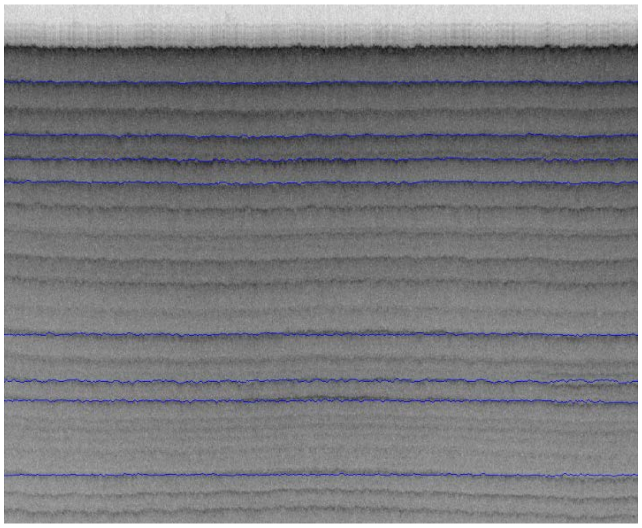

Figure 3: Iterated Conditional Modes

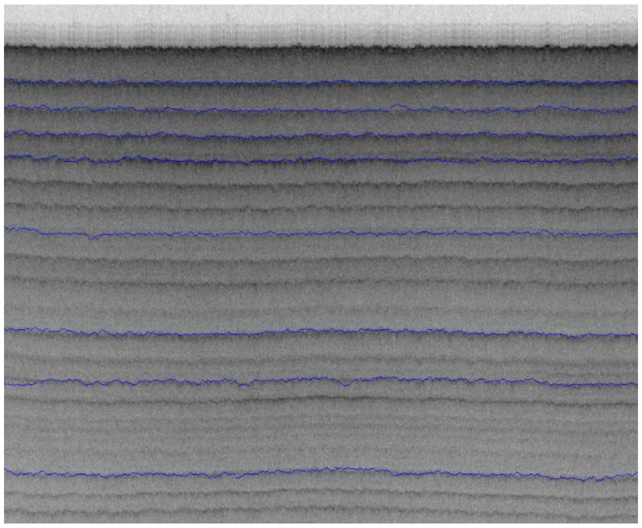

Figure 4: Gibbs-based simulated annealing

Our ongoing work is applying this new technique to the much more challenging problem of locating internal (annual) layer boundaries in ice and snow. Our approach in [3] removes the restrictive assumption of [1] that there are a small number of layers, but still requires knowing the number of layers ahead of time. This is a restrictive assumption because it involves solving the challenging problem of model selection: a model with more parameters (layers) will always fit the data better, but will also overfit, yielding a result that is statistically meaningful but not useful in practice. We are currently investigating extending our model to use Reversible Jump MCMC [9] as a means of addressing this problem.

9. Snow Radar Imagery

Our efforts to accurately identify multiple snow features in polar radar imagery, includes a statistical graphical model using both local (iterated conditional modes) and global (simulated annealing

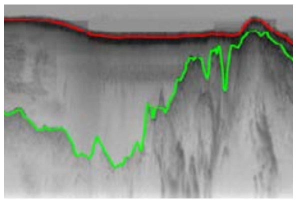

Figure 2: Results on three sample echograms. Each pane includes the hand-labeled ground truth image *(top-left)*, the output of ref. [1] *(top-right)*, and then our output *(bottom)*.

[10]) techniques shown in figures 3 and 4 for snow layer

Table 2: Overview of Compute resources in XSEDE, IU, and KU.

Name	Site	Peak Tflops	CPU Type	CPU speed	# Cores	# Nodes	Cores per Node	Memory
Stampede	TACC	9600	Intel Xeon E5-2680	2.7G	102400	6400	16	200TB
Comet	SDSC	2000	Intel Xeon E5-2680v3	2.5G	47616	1984	24	247TB
SuperMIC	LSU	925	Intel 64	2.8G	7200	360	20	22TB
***Gordon**	SDSC	341	8-core Sandy Bridge	2.6G	16384	1024	16	64TB
Darter	NICS	248.9	Intel	2.6G	23168	724	32	45TB
Trestles	SDSC	100	8-core AMD Magny-Cours	2.4G	10368	324	32	20TB
Blacklight	PSC	36	Intel Xeon X7560	2.27G	4096	256	16	256TB
***Karst**	IU	85.1	Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz	2.6G	4,096	256	16	8GB
***Bigred2**	IU	1006	AMD Opteron(tm) Processor 6380	2.5G	21,824	1020	32	43648 GB
***Drebber**	KU		AMD Opteron 2.4G	2.4G	128	32	4	256GB
***India**	IU	11	Intel(R) Xeon(R) CPU X5570 @ 2.93GHz	2.93G	1024	128	8	3TB
***Delta**	IU		Intel(R) Xeon(R) CPU X5660 @ 2.80GHz	2.8G	192	16	12	1.3TB
***Bravo**	IU	1.7	Intel(R) Xeon(R) CPU E5620 @ 2.40GHz	2.4G	128	16	8	3TB
***Echo**	IU	2	Intel(R) Xeon(R) CPU E5-2640 0 @ 2.50GHz	2.5G	192	16	12	6TB

determination. In iterated conditional modes [9], an initial estimate of the labels uses a deterministic “greedy” strategy to determining which labels gives the largest decrease in energy function; this process is repeated until convergence. In simulated annealing [11] a temperature parameter is reduced while maintaining current and neighboring variables. In each iteration, the energy is calculated for the current and neighboring variables; if assigning a value to the variable is an improvement, the algorithm accepts the assignment and updates a new current assignment. Otherwise, it accepts the assignment with some probability.

10. Conlusion

In this paper we have first, shown that the performance improvement of the multi-look domain processor and the use of modern hardware resulted in an overall performance improve this performance is done on a single core. When applying the same improvements to Quarry a supercomputer on which the analysis is to be run we only obtain a factor of 5. In our future work we will be investigating the increase use of parallelism in the workflow pipeline and the utilization of multicore features. Use of multicore may be hindered by too little memory on some of the machines.

Second, we have provided a new algorithm, that accurately identifies multiple snow features in polar radar imagery, includes a statistical graphical model using both local and global methods. The algorithm is tested on analysis of ice-sheet boundaries and snow radar imagery.

We have identified that we can predict to use 1.5M core hours on Gordon. We like to additionally apply for 100K core hours split between our efforts to improve the feature detection and the MLTPD algorithms.

11. ACKNOWLEDGMENTS

We like to thank XSEDE for their generous use of the startup allocation to obtain the performance on XSEDE resources. We like to thank Indiana University for the use of Big Red 2, Karst, and Quarry (the later is now transitioned to University of Kansas). We like to thank University of Kansas for the use of Drebber. Other resources used are part of FutureSystems part of the School of Informatics and Computing. We like to thank Hyungro Lee and Badi Abdhul-Wadi for their help on cloudmesh.

12. Resource Details

13. REFERENCES

[1] D. J. Crandall, G. C. Fox, and J. D. Paden. (2012, Layer-finding in Radar Echograms using Probabilistic Graphical Models. Available: http://vision.soic.indiana.edu/wp/wp-content/uploads/icpr12-ice1.pdf

[2] . *Cresis Web Page*. Available: https://www.cresis.ku.edu

[3] S. Lee, J. Mitchell, D. J. Crandall, and G. C. Fox, "Estimating bedrock and surface layer boundaries and confidence intervals in ice sheet radar imagery using MCMC," pp. 111-115, 2014.

[4] G. von Laszewski, A. Younge, X. He, K. Mahinthakumar, and L. Wang, "Experiment and Workflow Management Using Cyberaide Shell," pp. 568-573, 2009.

[5] G. von Laszewski, F. Wang, H. Lee, H. Chen, and G. C. Fox, "Accessing multiple clouds with cloudmesh," pp. 21-28, 2014.

[6] G. von Laszewski, *A parallel data assimilation system and its implications on a metacomputing environment*, 1996.

[7] G. von Laszewski, "An Interactive Parallel Programming Environment Applied in Atmospheric Science," in *Making Its Mark, Proceedings of the 6th Workshop on the Use of Parallel Processors in Meteorology*, ed Reading, UK, 1996, pp. 311-325.

[8] L. M. H. Ulander, H. Hellsten, and G. Stenstrom, "Synthetic-aperture radar processing using fast factorized back-projection," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 39, pp. 760-776, 2003.

[9] P. J. Green, "Reversible jump Markov chain Monte Carlo computation and Bayesian model determination," vol. 82, pp. 711-732, 1995.

[10] S. Kirkpatrick, J. C.D. Gellatt, and M. P. Vecchi, *SCIENCE,* vol. 220, Number 4598, 13 May 1983 1982.

[11] J. Besag, "On the Statistical Analysis of Dirty Pictures," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 48 No. 3, pp. 259-302, 1986.