Pre-Hardware Optimization of Spacecraft Image Processing Software Algorithms and Hardware Implementation[1]

Semion Kizhner, David J. Petrick, Thomas P. Flatley, Phyllis Hestnes, Marit Jentoft-Nilsen, Karin Blank
National Aeronautics and Space Administration
Goddard Space Flight Center
Greenbelt Road, Greenbelt MD, 20771
301-286-7029
skizhner@pop500.gsfc.nasa.gov
tflatley@pop700.gsfc.nasa.gov

Abstract–Spacecraft telemetry rates and product complexity have steadily increased over the last decade presenting a problem for real-time processing by ground facilities. This paper proposes a solution to a related problem for the Geostationary Operational Environmental Spacecraft (GOES-8) image data processing and color picture generation (GOES-8 application). Although large super-computer facilities are the obvious heritage solution, they are very costly, making it imperative to seek a feasible alternative engineering solution at a fraction of the cost. The proposed solution is based on a Personal Computer (PC) platform and synergy of optimized software algorithms and reconfigurable computing hardware (RC) technologies, such as Field Programmable Gate Arrays (FPGA) and Digital Signal Processors (DSP). The solution involved porting the GOES-8 application from its Silicon Graphics Inc (SGI) Workstation/UNIX platform, making minor platform specific changes to the GOES-8 application (so that it runs on the PC), benchmarking the various code segments, and implementing the most compute intensive functions in hardware. After pre-hardware optimization steps in the PC environment, the necessity for using RC hardware implementation for bottleneck code became more evident. The problem was solved beginning with the methodology described in [1], [2], [3], and implementing a novel methodology for this application. The PC-RC interface bandwidth problem for the class of applications with moderate input-output data rates but large intermediate multi source data streams has been addressed and mitigated. This opens a new class of satellite image processing applications for bottleneck problem solution using RC technologies. The issue of a science algorithm level of abstraction necessary for RC hardware implementation is also described. Selected software functions already implemented in hardware were investigated for their direct applicability with the intent to create a library of RC functions for ongoing work. A complete class of spacecraft image processing applications development using re-configurable computing technology to meet real-time requirements, including methodology, performance results and comparison with the existing system, is described in this paper.

TABLE OF CONTENTS

INTRODUCTION

The nominal telemetry rate of the GOES-8 imager instrument is 2.22 Mega bits (Mb) per second (Mbs) and the ground processing system receives information sufficient to produce one color picture in 75 seconds. The GOES-8 application must be capable of processing the data and of producing the largest size output color picture in fewer than 75 seconds, or before the information arrival of the next image telemetry data is complete. *This 75-second threshold is the Real-Time (RT) processing criteria.* The GOES-8 operational application currently takes 291 seconds to generate a large color picture from telemetry. This is approximately 4 times greater than the real-time requirement of 75 seconds.

Simple real-time processing of GOES raw telemetry data has been accomplished using data captured by an antenna on a school rooftop and processing a telemetry band, at a time, out of the 5 bands. There are reasons why the generation of a large color picture takes so long. This is

[1] "U.S. Government work not protected by U.S. copyright."

because the GOES-8 application is scalable and can generate small, medium and large color pictures. The moderate volume of GOES-8 data required to generate a single output color picture product is comprised of 12.5 Mega Bytes (MB) source telemetry (four infrared bands and a visibility band), a 10.55MB prebuilt color map and a prebuilt 0.425MB navigation file. Some intermediate data, like sun angles, is derived during processing, bringing the basic unit operations data volume to 25MB. Furthermore, this moderate volume of data is expanded to 400MB within the GOES-8 application during intermediate and complex science data processing required for the large color picture generation. The moderate telemetry volume expansion by a factor of 30 and the associated computation complexity phenomena are at the core of this application performance problem.

NASA's Dr. Dennis Chesters, the GOES-8 Principal Investigator, and Marit Jentoft-Nilsen from the GOES-8 application development and operations team *formulated the problem:* in order to meet the real-time processing criteria the existing GOES-8 application's run time that comprises all processing required to produce an output color picture, needs to be improved by at least a factor of four. They also provided the GOES-8 application's Interactive Data Language (IDL) (Trademark of Research Systems, Inc.) source code, input test data set and the native SGI Workstation/UNIX platform timing performance benchmarks (Section 5.0 Table 2). It was assumed in this project that all production input data will be similarly provided by the GOES-8 operations center on the native Workstation/UNIX platform, accessed over the Internet by the proposed solution PC/RC platform, and that the output will be directed to the same native Workstation/UNIX platform over the Internet. Whilst, when the problem is solved the solution can be demonstrated in the development laboratory using real time telemetry.

The methodology that was used to solve this problem and the solution comprise the subject of this paper.

1.0 APPLICATION RUN-TIME COMPLEXITY AND THE SOLUTION METHODOLOGY

The phenomenon of moderate telemetry volume expansion during science data processing is not specific to the GOES-8 image data processing. This expansion often originates in science data processing required to generate a complex product. In the case of the GOES-8 application such a product is a large color weather picture obtained by processing telemetry 900x750 pixel multi band images. The additional processing of GOES-8 spacecraft raw telemetry involves computing sun angles for each data pixel, as well as this small telemetry grid up-scaling to 1800x1500 pixels (medium picture size) or to 3600x3000 pixels (large picture size) and the associated with it interpolation, filtering and smoothing algorithms that comprise the science data processing complexity.

Spacecraft telemetry rates and telemetry product complexity have steadily increased over the last decade. Generating complex products from even moderate input data volumes results in large intermediate data volumes. When a large intermediate input-output data volume is coupled with multi band and color nature of data and intensive computations it becomes the core of a data processing application performance problem.

1.1 Problem Complexity and Solution Goal

In the GOES-8 application, there is a need for a dual exchange between the PC and RC board of 400MB of intermediate data, and concurrent processing on the RC component of an equivalent of 1080 Millions of floating point operations (MFLOPS) within 30 seconds. This presents a significant challenge to a low cost platform and requires methodology considerations at the project outset.

The goal is to solve the problem with minimum resources. This, for example, can be accomplished by using a DELL Dimension XPS T550 PC on a Windows98 Operating System with 768MB SDRAM (three 256 MB PC1 100MHz SDRAM ECC memory modules), a Texas Instrument Inc. (TI) DSP board TMS320C6701 and an FPGA StarFire board from Annapolis Micro Systems Inc.

A single inexpensive PC platform selection is motivated by the interest in concentrating the processing intelligence, in view of plans for extending the task from a ground platform to spacecraft. There are also available inexpensive FPGA and DSP boards for the PC utilizing the Peripheral Component Interface Bus (PCI) [5]. Furthermore, in addition to the solution goal the main objective is to develop in-house capabilities to apply this solution to a wide number of other applications. The solution's methodology depends on a few basic considerations that stem from this goal and main objectives.

1.2 Methodology Basic Considerations

The origins of the large data volumes and intensive computations were introduced above. It becomes apparent that there is need for large memory resources, both on the computer platform and its RC components, to handle large volumes of data. There is consequently a need for fast data transfer interfaces between the PC and RC board, as well as between RAM memory and CPU on the PC, and external memory and data paths on a DSP board. It was known that the PC/PCI/FPGA board interface is utilizing 70% of the PCI bus bandwidth and that the corresponding and advertised DSP board ideal throughput is 160 Mbs (the practical IO bandwidth for the PC-DSP PCI interface appears twice less than this number). Because the RC boards carry small external memories there is a need to partition the problem and associated data PC-RC streams into IO blocks to satisfy memory limits. This, however, requires careful sizing of the IO block (n) and the number of IO transfers (k) to optimize the practical throughput and,

in turn, requires a top Application Programming Interface (API) level synchronization mechanism between the PC and RC components. The RC technology state-of-the art presently has limitations in these areas, namely the FPGA board allows fast PCI bus and on-board IO operations but lacks serial floating point arithmetic (FPA) while the DSP board provides the FPA but has smaller IO bandwidth. This contributes to the problem complexity and requires pre-hardware optimization efforts in order to use the RC board advantages. New technologies allowing RC access by using a PC RAM slot and floating point arithmetic libraries for FPGA could soon provide a better RC option.

1.3 Methodology Roadmap

The proposed engineering solution methodology roadmap was first to configure an IDL development environment on a PC platform, convert the application from its native SGI/UNIX platform to a lower cost PC/Windows platform (Windows Operating System is a Trademark of Microsoft Corporation) and make it work on the new platform in software and generate color pictures which are comparable to those produced on the SGI/UNIX platform for the same test set of input files. The workable PC IDL code application software is then optimized to achieve comparable to the SGI/UNIX platform or better run-time performance and determine the PC bottlenecks. And finally, the PC IDL software code bottlenecks are implemented in high-speed, reconfigurable, computing hardware to achieve real-time performance. The selected RC hardware must make use of floating point arithmetic required by the bottleneck science algorithms. Because IDL is not an open source code environment the conversion of the entire GOES-8 application into RC hardware is not feasible since it would require reconstruction of many complex IDL algorithms. Following are the few important steps along this methodology roadmap implementation.

1.4 Methodology Critical Steps

Configuration control of the preempted GOES-8 application IDL software version V2 and the newly procured IDL development system (for Windows98) Version 5.3.1 for the duration of this project was essential to this project success. It is a mature application that has been used in operations for a long time. This allowed freezing the code and its IDL development system version for the duration of this project (a few changes were made to the application code required due to platform conversion).

This paper describes the methodology of pre-optimizing the application software on the new PC platform and seeking out bottlenecks before bottleneck code implementation in RC hardware such that the native platform performance benchmarks are improved first. This approach allows to develop insights into application internals and specifics before attempting a front large-scale implementation of code in hardware.

The PC-RC large intermediate multi source data streams IO problem has been addressed and mitigated. This problem is one of the reasons why some similar applications were previously implemented in full in RC hardware, and at greater costs. In such an application a moderate data rate single stream input and output is handled by the RC hardware, as well as all the computations. Exchange of large multi source intermediate data streams between host PC and RC hardware is avoided at the expense of a very complex implementation of an entire application in RC hardware.

Searching for the bottleneck code segments and only their RC hardware implementation, rather than the entire application, is the main theme of this paper. This methodology is based on the solution to the PC-RC large intermediate multi source data streams exchange problem.

Making the converted application run on the new PC platform and validating the software product correctness before hardware implementation tools was important in avoiding later problems. This validation was done by running the application on both platforms for the same set of operational input files, and comparing the resulting pictures. The native platform application may evolve and, because of resulting picture file lossey compression, a slight difference in the product pictures is expected and acceptable.

1.5 Initial Observations and Lessons

Just porting the application from its native operational platform to the PC platform and limited pre-hardware optimization improved the basic performance of the application and provided new insights into the problem, namely:

a) The PC platform production of the small and medium size images (the scalability of the problem is described below in Section 2.0) is faster by a factor of two in comparison with the native platform and is already meeting the real-time criteria.

b) There was a performance loss in comparison with the native platform benchmark for the generation of the large color picture using the initial PC memory configuration of 128MB. It was determined that the PC memory size is a significant factor in this application performance. The PC platform memory was then upgraded to its maximum allowable 768MB memory configuration including the PC BIOS upgrade. This greatly improved the performance.

c) It is interesting that the native platform general performance bottlenecks shifted on the PC to other, less expected places in the application. Upgrading the PC platform memory to meet most of the application's requirements and splitting the large picture generation bottleneck code into 16 segments led to improving this case difficult benchmark by a factor of two, and reaching the real-time criteria performance goal within 50 seconds. However, the 768MB memory and BIOS upgrades alone were not enough to achieve this result. The original process was still slow with this maximum memory configuration. Only when splitting was introduced did we reach the 120

seconds result. This splitting also contains elements of classical loop unrolling to economize on index processing and the resulting data type was of single precision floating point as opposed to double. This type change and its implications are further described in Section 6.
d) Local optimization of functions may degrade the application overall system performance.

The first observation (a) is quite natural: the newer PC platform has a faster Pentium III processor and there is a possible gain in computational performance in comparison with the older workstation. *The second observation (b)* can be attributed to the PC/RC platform memory size and configuration limitations. Since the processor is accessing memory far more than any other device, and a shortage of Random Access Memory (RAM) is definitely causing more access of virtual memory on hard disk, the memory size and access speed are important performance parameters. There are subtle issues of platform computer cache memory level(s) size, the number and size of computer/RC board registers and operating system memory management, as well as Input-Output (IO) drivers' quality and data bus bandwidth for PC-RC components data flow. For example, each of the PC platform's internal data storage levels *Disk-RAM–Caches–Registers* may differ roughly in data access speed by a factor of 4. *The third observation (c)* was remarkable in the respect that the shift appeared in a segment of code that allowed straightforward hardware implementation and performance gain by a factor of ten and consequently allowing to meet the real-time requirements. The analysis of this segment of code also revealed the significance of platform's memory hierarchy and its utilization by the bottleneck code processes. The fourth observation (d) is subtle. It is important that optimization of a selected code function in local execution mode (local optimization) does not degrade the entire application performance (application global optimization). A code function may be forced into local minimum, for example, by allocating to the function more system resources (such as memory) and this may degrade the global system performance. Sharing the task analysis and up to-date results with the application operations team ensured that improvements on the native platform were made as a result of this effort on the new development platform. As a result of this approach the application was also moved from the native UNIX platform to a newer interim UNIX platform and a new set of benchmark timing performance (UB2) was provided by the GOES team (Section 5.0 Table 2) as this work was in the hardware implementation phase.

2.0 THE HERITAGE APPLICATION OVERVIEW

A nominal run of the GOES-8 heritage application consists of processing a set of telemetry-based input files (supplied by preceding levels of operations) and producing color pictures (end-product) used for weather prediction. The typical run produces a single color picture using a sub-set of input files of origin {title, nav, map, vis, ir2, ir3, ir4, ir5}. Each telemetry file name begins with a timestamp prefix in the format of YYMMDDHHmm that uniquely identifies each subset. The rest of a file name is a fixed root G8I (GOES-8 Visibility and Infrared bands) and followed by a suffix 01 for 'vis' files, 02 for ir2, 03 for ir3, 04 for ir4 and 05 for ir5 bands. For example, an ir2 file that originated on February 12, 2000 at 10 hours and 02 minutes GMT time has a name 0002121002G8I02.tif. The file name timestamp must be within the specified for the run range of telemetry times. The files that comprise this processing unit are

{0002121002G8I01, 0002121002G8I02
0002121002G8I03, 0002121002G8I04
0002121002G8I05}.

These 'tif' type files and the accompanied background color map and navigation data files are described in more detail in the following sections.

2.1 Application Scalability

The problem-scaling factor is specified in the application top-level procedure as a software switch and determines the size of the output color picture. This factor is the main contributor to the application run time computational complexity. The basic size of the output color picture is 900x750 pixels (switch value "small" equal to telemetry image size) and 59 seconds are required to run it on the native platform (UB1, UB2 Section 5.0 Table 2). The picture "medium" size is 1800x1500 and the large picture size is 3600x3000 (switch value "large") and requires expansion of the basic telemetry image by a factor of 16. Although the first benchmark value for the small picture generation on the native UNIX platform (UB1) is already under the real-time requirements criteria of 75 seconds, the present benchmark performance for the medium size picture is 100 seconds and for the large picture 291 seconds, or 4 times larger than desired, and needs at least that much an order of improvement for the application to meet the real-time processing criteria. A typical production heritage run can be interpreted as a sequence of independent units each processing a file set as described above. Each file is read into RAM and appropriately scaled into intermediate arrays whose size is depending on the hard-coded scale factor.

2.2 Application Resources Requirements

For large color picture generation, the application's largest bottleneck code segment is operating with intermediate data in the form of 6 large arrays. One of these arrays is of double precision floating point type (8 bytes) and 5 arrays are of single precision floating point type (4 bytes). Therefore the input arrays memory requirements can be estimated to be 302.4MB. The three resulting arrays are a subset of the input, but change type from input single to output double precision floating point type during a bottleneck recomputation. This requires an additional 259.2MB of RAM increasing the minimum RAM requirement to 561.6MB. There is always a need for some memory depending on system internal operations and this

can only be found experimentally by monitoring the computational process memory allocation. It was established experimentally that the GOES-8 application required some 720MB of heap (dynamic) memory for the problem 'large' case factor. However, even upgrading the PC/RC platform to 768 MB RAM did not solve the problem completely, but rather clarified the need to further analyze the application and find areas of its possible implementation in RC components on the new platform using parallelism paradigms. There is no doubt that 1GB of memory and a 1GHz PC could solve this application performance problem. However, the obvious solution of another super computer is not acceptable because it would take years before it can be made ready for space flight. Furthermore, there is still a need to free a powerful PC resource to perform intelligent processing most of the time and move the routine intensive computations to an RC component.

2.3 Application and Problem Partitioning for RC Implementation

There are a few avenues of achieving parallel processing benefits using FPGA or DSP RC technologies. They may utilize different aspects of an application depending on its Computational, IO and Interface requirements. To depict the application natural avenues of possible parallel re-configuration and problem partitioning we analyze the application critical aspects and provide the model of this application structure and data processing.

2.3.1 Computational Aspects

The computational aspects of the PC/RC platform include the PC platform and RC software or firmware processors and comprise:

- PC processor computational speed
- PC internal memory (RAM, caches, registers) sizes, memory management by the Operating System , and overhead of IO drivers
- Algorithms pre-hardware optimization
- RC board computational speed
- RC components memory types and sizes
- Existence in the application of independent bottleneck code segments allowing segments parallel processing implementation in hardware
- Image array interpretation as a vector for pixel-by-pixel processing.

2.3.2 Auxiliary Storage and IO Interface Aspects

Auxiliary storage and IO data volumes and throughput aspects may be the most important limitations in this project. They are as follows:

- Moderate size operational telemetry file sets at remote site and product destination site allow the use of Internet for file level transfers
- A PC/RC platform has adequate disk space size for autonomous local IO operations
- A data processing unit is an independent file set on disk that is fully loadable into a PC RAM to reduce disk IO during processing
- A large volume of intermediate multi source data is created in this application.

The data originates in different source files and arrays, presenting a multi source aspect that requires multiple fast IO transfers over the PCI bus between the PC and RC components mainly because RC boards have small external memories. For this reason the initial RC implementation was reduced to a subset of the bottleneck code and 259.2MB of data. The sustained data throughput supported by the available DSP board was found experimentally and is close to 80Mbs. This provides an estimate for the time required to transfer the data from the PC to the DSP and back as 26 seconds. If the bottleneck computation on the DSP can proceed concurrently with the data transfer we can achieve the reduced bottleneck processing under 30 seconds using a DSP board. The FPGA board has advanced IO capabilities and utilizes 70% of the PCI bus bandwidth.

2.3.3 Data Multi Source Aspects

The solution to the data multi source and large IO volume aspects of the listed above domains may, in turn, be another strong reason for a bottleneck code, as opposed to full code, implementation in hardware. Full code implementation relies on a single moderate rate stream IO, and thus narrows the satellite image processing application domain. For example, the PC-RC IO, over the currently fastest available interface, the Peripheral Component Interface Bus using CPU controlled or Direct Memory Access (DMA) IO, significantly depends on the application IO data nature. A single large data stream is the ideal case of DMA/PCI IO mechanism. A multi source case when the RC target data originates in different arrays on the PC presents an IO performance problem and RC memory operand access problem because the operands from different data sources are widely separated in RC external memory locations. This is not the DMA/PCI channel problem, but it is based in the overhead of multi source DMA initiation or in pre-building a single thread from the multi source data both on the PC and the RC components. This multi source IO problem has also been solved in this project. The chosen approach to address this application multi source IO problem can also be used for a wider class of satellite image processing applications. The solution is illustrated in Section 7.2.

2.3.4 Application Model

The domains described above and their multiple interfaces are depicted in the following processing model of the GOES-8 application. The heritage processing deals with a fraction of the nominal telemetry stream at a selected rate of 0.2 Mbps resulting in a unit of data each 15 minutes or 10 times less the science maximum capacity rate of the GOES-8 spacecraft. By re-organizing the heritage sequential processing of independent processing data units

$$\{D1, D2, \ldots, Dj, \ldots\}$$

and the associated sub-processes of, say, three bottleneck code segments {P1, P2, P3} (performance of which is also a function of the data Dj size and content)

{ {D1,P1,P2,P3}, {D2,P1,P2,P3},... {Dj,P1,P2,P3}, ... }

into problem partitions

(Dj,P1) (Dj,P2) (Dj,P3), (where j=1,2,...,k)

the bottleneck processing can be implemented in parallel on an FPGA or DSP (or more than one RC board – an FPGA and a DSP pair), and can solve the run-time performance problem. This, in turn, would allow support of real-time image processing at the present rates and also at increased rates of science data by a factor of ten. The combined power of PC and RC system will keep the auxiliary storage cumulative input data set small most of the time.

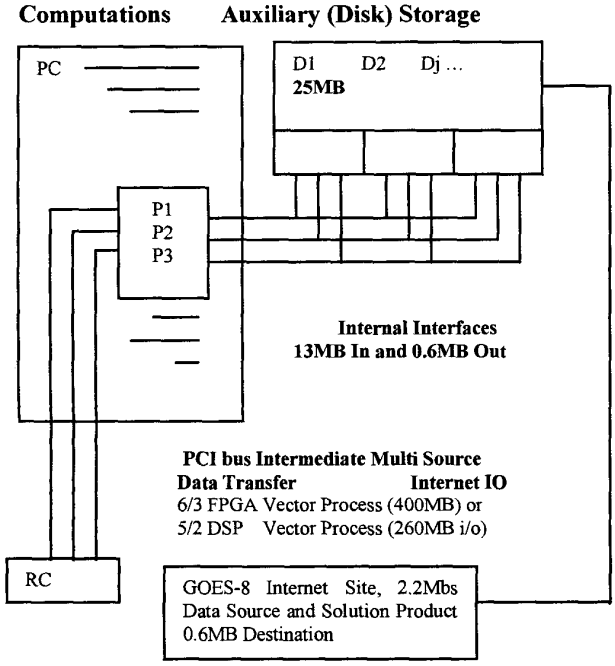

Figure 1. GOES-8 Application Model

2.4 Data Structures and Volumes

The 'nav' file, whose size is 0.428 MB, contains relevant geographical information - a sequence of pairs of latitude and longitude for the 361x301 grid of the spacecraft image pixels preceded by a two-word header specifying the array dimensions. The map file (a fixed prebuilt file) stores a pre-selected terrestrial map on which the spacecraft images are superimposed to get a color picture. The map file requires 10.34 MB of disk space. The input files ir2, ir3, ir4, and ir5 store spacecraft images in four infrared bands and require 0.659 million bytes (MB) of disk space each. The 'vis' or ir1 file stores the visible band image and requires 10.548 MB of disk space. All input files except the nav file are Tag Image File (TIF) format files. A unit processing single output color picture is a 0.074MB (small color picture) to 0.201MB (medium color picture) to 0.600MB (large color picture) file. This file is in a lossey-compressed file in Joint Photographic Experts Group (JPEG) format. The pre-hardware implementation timing tests performed on the PC platform are summarized in Section 5.0 Table 2. The application porting and a few initial enhancements to the application software were based on experimental insights into IDL and the application IDL code, and are described next.

3.0 Porting the Application

The reconfigurable computing technology is just maturing and is in need of a proficient application, which spacecraft image processing can provide. At this time, implementation of a telemetry processing application in hardware necessitates careful pre-hardware implementation steps, such as selecting a platform for which inexpensive RC hardware exists, porting of an existing application (such as the GOES-8 color picture generation application) from its native platform to the new platform, defining and optimizing algorithms, and selecting software programming languages conducive to full or partial translation of a software application into hardware. These steps are described below on the precedent of the GOES-8 application, especially the porting of the application from its native SGI Workstation/Unix platform to the PC/Windows/RC platform.

3.1 IDL Platform Dependent Issues and Their Resolution

A PC platform running the Windows operating system and the same version of IDL as the native platform was selected as the new platform. There exists a series of inexpensive RC hardware boards and associated tools for the PC platform. Firstly, the GOES application IDL source code and test data files were ported (using the Internet file transfer protocol) from its native platform to the PC/Windows development platform. A free IDL demonstration package for the PC/Windows development platform was obtained from the vendor and rapidly evaluated to ensure the application feasibility on the new platform. The application software and data system initial study and evaluation on the PC/Windows platform were conducted. This resulted in proving the feasibility of reimplementing the application on the PC/Windows platform running IDL.

Secondly, two full IDL development systems Version 5.3.1 for the PC/Windows platform were procured, including system maintenance and technical support. Both the IDL system and the application were rapidly modified to allow full compilation of the project and its successful run on the new platform.

The porting of the GOES-8 IDL application from its native Workstation/UNIX platform to a PC/Windows/FPGA/DSP platform carries an associated and expected slew of

platform depended issues. The few IDL system platform dependent differences are mostly due to differences between the two platforms' operating systems. The NASA and IDL vendor teams quickly resolved these differences. For example, the IDL function SPAWN is using platform operating system commands to spawn a concurrent process. Since the operating system commands are different they must be re-coded. File name and directories' paths specification and convention differences require change in parsing algorithms and code. The PC is also a so-called "little-endian" machine that swaps the two bytes in a 2-byte word when reading input binary files from disk unlike the UNIX Workstation. This must be addressed when accessing files in other than 'tif' or 'jpeg' format, like the 'nav' binary format file. The native SGI Workstation/UNIX platform is using the IEEE-754 format for handling integer and real numbers, as does the new PC platform. The application does not have the new platform computer identification 'x86' in its internal table and defaults to an SGI system representation of integers and floating-point numbers after issuing a warning message. However, both platforms use the same IEEE-754 standard for integers and real numbers representation, and this warning can be ignored. The changes induced by the platform differences are fully described in the following section.

3.2 Source Code Modifications and Compilation

Platform-dependent modifications to the application source code were required in order for the PC IDL project to successfully compile and complete a full run. IDL project "GOES" was created on the PC/Windows98 platform using the modified UNIX IDL source code modules. The differences between the UNIX and the Windows98 operating systems and the IDL Version 5.3.1 specifics and its implementation features for Windows were determined. The corresponding changes to the GOES project source code were then made in order to eliminate compilation errors. For example, the two platforms have a typical *CR-CRLF* text line delineation incompatibility. The UNIX forward slashes '\' in directory specifications were changed to Windows notation '/' to allow correct parsing of directory and file names. The syntax of the SPAWN was changed to execute a DOS batch file. The IDL option /SWAP_IF_LITTLE_ENDIAN was applied to the IDL OPENR, which solved the problem in handling the navigation data file.

Compilation of the complete GOES project on the PC/Windows98 platform was a milestone, as was the first run of the project on the new platform.

The top operational procedure was updated with IDL timing analysis debugging statements and the *help, /memory* statement. This allowed precise determination of the maximum heap memory required by this application. In summary, six source code modules were changed on the PC/Windows98 platform and the depth of required changes was minimal.

The GOES-8 application comprises 98 procedures, functions and definition software files written in IDL. These are using most of the features provided by the IDL development environment. The model of the entire application was provided above in Figure 1. It depicts the application structures, which are susceptible to parallel implementation in hardware.

3.3 First Successful Run

The first successful program run on the PC platform was performed in two weeks after the PC platform IDL development system arrival and installation. All of the changes that were previously made effectively worked. This run was made using the ported application for its default output picture size of 1800 x 1500 (medium) and ten available sets of input files. All 10 resulting color pictures were written to the correct output directory in 'jpeg' format. The pictures were immediately verified by just viewing them while formal verification tools were under development. Performance timing information was recorded for each of the 10 files (Section 5 Table 2). The average run time for a single loop to produce an output medium size image was 80.21 seconds. The operation that consumed the most time was 'calc v,s,b with vis'. This operation averaged 36.27 seconds. The IDL function color_convert took an average of 12.48 seconds varying from 6 to 20 seconds for a large image generation (this processing is evidently data dependent). Due to their hardware-compliant algorithms, these two operations will be the first candidates for RC implementation. After the PC memory was upgraded from 128 to 256 MB the problem medium size case run time was reduced to 36 seconds and meets the real-time requirements. However, achieving real-time performance for the largest size output picture remained a challenge. The solution for all sizes of output color pictures is described as follows.

3.4 Development of Tools

A complement of tools was explored and developed for this conversion project. The tools fall in two main categories - traditional tools used in software development (TSW) and tools for RC hardware implementation (TRC). These tools for input files and output color picture analysis and computational performance studies are described in the following subsections.

3.4.1 Data Analysis and Software Debugging Tools

The publicly available software tool Hexview.exe was used to analyze the navigation file (nav) structure. This resolved the PC platform "endian" problem.

The structures of image files of type 'tif', 'tiff' and 'jpeg' were analyzed using query tools provided by IDL and printing the returned file structure variable. These formats' latest standards were retrieved from its holders' Internet site and used in tool development.

An IDL procedure was developed to compare the content of two 'jpeg' format image files. This procedure takes the names of two files to be compared as input parameters, parses the file names to determine the file type 'tif' or 'jpeg', reads them into corresponding arrays using IDL read procedures, and compares the contents pixel by pixel, and returning the count of mismatches for differences exceeding a threshold. The comparison was done before and after compression. The IDL utility functions were used to determine the input files' data types (integer, floating and pixel value size byte, 2 bytes, 4 bytes, 8 bytes) and indicators of lossey file encoding.

3.4.2 Tools for Memory and Timing Requirements Studies

It was quickly realized that the PC platform memory size is a critical factor in the application timing performance. However, it was difficult to determine the exact memory requirements for the GOES-8 application. IDL provides a powerful function for inquiring the memory heap state at any given point in the IDL code, namely:

help, /memory

This returns and prints information that was decisive in the selection of optimal memory size for the new platform and a platform that allows to be configured with this memory size. IDL also provides the standard tools to inquire about current *system time* and elapsed time between any two points in the application code. It was also noticed that when initially internal memory was not sufficiently large, computational references to the part of virtual memory on the hard drive overloaded the system. Memory upgrade would also prolong the hard drive lifespan. Although platform memory size is significant in timing performance of an application, the timing performance is a more complex function of platform memory size, PC cache size, operating system management of memory and especially its stack and heap and other factors. The PC host and RC target C-code modules' timing was performed using ANSI C standard timing functions. However, they were found to be inadequate, and a timing function to accesses the Pentium time stamp counter was developed in assembly language.

3.4.3 Tools for PC Host and DSP Target Modules Debugging

There is no standard output feature in IDL V5.3 for the external host module. This issue was resolved by introducing in the host module a text file to store debugging messages and analyzing them off-line at test end. Similarly, there is no ready debugging mechanism outside the TI Code Composer Studio. The issue of debugging the DSP target module that implements the IDL bottleneck code was resolved as follows. The few lines of IDL bottleneck expressions were copied from the large GOES-8 IDL source code and used in a small test IDL project. These lines were then preceded by a few more lines of array initialization to known values using IDL functions. This small project was augmented by a few IDL code lines to display initial values from arrays' first five and last five pixels before and after computation. These then comprise the truth test data. Next, the IDL-host module-DSP target module calling sequence line is introduced into this test project replacing the IDL bottleneck expressions and is run again. This yields results from DSP target module computations, which are compared manually with the truth data.

3.4.4 Tools for Array Pointer Validation Across Different Programming Environments

A stand-alone program in C++ was developed to determine the actual pointer handling issues across the IDL-Host C++-DSP target code handling. The issues of pointer assignments for initialization, pointer assignment expressions needed for step-down into vectors and mixed type pointers handling were analyzed and solved using this program.

3.4.5 Tools for Color Picture Validation

The 10 input file sets were used throughout this project. These files were then given back to the native platform operations team and processed there. The pairs of corresponding pictures obtained on both platforms were compared using the tools described above in Section 3.4.1.

4.0 PERFORMANCE TIMING METHODOLOGY

The task of an application run time optimization is difficult when software has already been implemented but not fully documented. However, there are some benefits in working with an existing, fully functional application such as the GOES-8 application. Existing application science algorithms and software code can be studied directly and timing tests can be readily performed in order to localize application performance bottlenecks. The following is the description of the methodology employed in the run time performance study of the GOES-8 application (also called here IDL GOES-8 project) on the PC/RC platform. The IDL GOES-8 project operands (any IDL code except user developed function calls) and the algorithms and options of major IDL code functions were studied. The affects on run time performance of minimum and minor changes to both IDL operands code and algorithms were also studied.

Timing studies have uncommon complexity since run time depends on many variables of different origin. Our interest is in the optimization of the GOES-8 application run time function T(p) – the run time required by the GOES-8 application to generate a single color picture product, specifically

$$T(p) = T(d, g, a, c, C, i, r, O) <= T_0,$$

where T_0 is the real time criteria threshold defined in the Introduction as 75 seconds. The variables are described as follows:

- telemetry volume (d)
- intermediate data expansion factor (g),
- host algorithms time complexity (a),

- PC computational complexity (c),
- RC computational complexity (C),
- PC-RC interface bandwidth (i),
- Internet Interface bandwidth (r)
- other contributing variables (O) like O=O(M), where M is the RC on-board external memory size.

T(p) not only depends on the numerical values of the parameters d, g, a, c, i, r, O but also on the nature of the source data. The data used in timing studies, with all these parameters fixed, may yield a ratio of 2 between T(p) upper and lower boundaries. For example, if an application comprises a single data comparison to a constant and a single result assignment depending on the comparison outcome, using a data test pattern of constant values to trigger the assignment will cause the application to run twice longer for this specific input data set. The standard approach on this time studying problem is to use carefully prebuilt benchmark data sets. However, this is impractical for this application and the timing tests data set was the test data set of 10 telemetry files obtained from the GOES-8 operations team and was considered as representative of the application standard input. These files were used to study the timing affects of arguments (d) and (g). To study the affects on the timing function of arguments (a) and (c) two mechanisms were used:

- *computing elapsed system time for IDL functional calls and*
- *investigating timing costs of basic IDL operands in large loops.*

These were used as the starting points in the attempt to improve the application timing performance, and resulted in determining the IDL algorithm and code bottlenecks. Further studies of the effects of variables (c) and (i) involved hardware implementation of the application IDL bottleneck code using RC boards. It becomes rapidly obvious that if selected for RC hardware implementation, bottleneck code's volume V=d*g requires time I(V, i) for the data to be transmitted and the results retrieved from an RC hardware board, then a simpler than T(p) relationship between I, c, C, Δt and T_0 determines the PC/RC bottleneck engineering implementation architecture, requiring that

$$c + I + \Delta t + C < T_0.$$

The PC-RC synchronization wait parameter Δt is described below.

Towards the goal of generating the GOES-8 color picture in real time on the new PC/RC platform, the obvious first objective is to make the application run correctly on the new PC platform. Towards this objective, the conversion effort was conducted from the UNIX to Windows operating system as was already described above. The second objective was to achieve on the new PC *platform a performance at least equal to the native platform benchmarks before RC hardware implementation.* Toward this objective, the PC platform memory was upgraded from an original 128MB to the PC maximum allowable configuration of 768MB and pre-hardware optimization steps were performed with steady improvement in the application's time performance. The third objective was to implement the PC remaining largest bottleneck code segment in RC hardware to achieve real time performance. For this, similar to the first and second objectives, the initial hardware implementation was debugged to produce a correct color picture without regards to timing performance. However, this came at a price of $C >> T_0$. Knowing that $I < T_0$, this then led to an effort to implement different than nominal pre-processing of source data in the host module before writing it to the DSP for intensive computation in order to attain DSP performance that yielded $C<<T_0$ or a ten-fold improvement. The pre-processing consisted of replacing separate IOs for all data sources with a single vector of quadruplets of source data pixels and a single IO block. Aggregating the multi source input pixels into quadruplets solved the multi source operand performance problem on the DSP. With this done, the T(p) becomes better than the software version and the remaining PC/RC implementation optimization involves attaining a better I by developing a better PC-RC PCI driver and to reduce redundancy in the data passed to-from the PC-RC board.

The IDL development system data types' specifics, as compared to other development systems, like Matlab (Trademark of MathWorks Inc.), were also determined. This is necessary for IDL code implementation in hardware because there are no cross-translators from IDL into Hardware Description Language (VHDL). However, it is known that some Matlab and C functions were implemented in a hardware library. Knowing the similarities and specific differences between IDL and Matlab can facilitate the IDL application partial implementation in hardware.

Automatic generation of the application software modules calling tree is required to analyze program time performance. The existing utilities to do this were analyzed and needed modification. Though this effort was started, pressed by schedule, we built the application's software calling tree manually. There are 76 procedure and function nodes in the tree and 25 end-nodes or leaves. There are also dormant nodes like that are not called at all and some that are not reachable on the PC platform, like the integer and floating point conversion function for VMS-PC number format conversions. The calling tree helps to evaluate the problem size and complexity.

The function call frequency table is required in order to find the modules, which are called most often. The tool to generate this table is provided within the IDL development system and the table can be used in selecting prominent functions for embedding them directly into code in order to reduce operating system overhead, while preserving code maintainability.

We have established a set of performance tracking tools such as timing different segments of code and monitoring heap activities for dynamic memory allocation and de-allocation. These include IDL and Windows directives as well as the timing points inserted by the GOES-8 application's development and platform conversion teams.

We have conducted an experimental study of the *time cost of basic IDL operations* in a loop of an average (medium) size pertinent to the GOES-8 application.

We have investigated the dependency of the application time complexity on the number of input data transactions (data driven component of time complexity). For example the filter construction operations "<" or ">" on an array (x) and scalar (1), say y=x<1, execution time greatly depends on the number of elements in array (x) that are larger than (1), because these elements are then replaced by 1 at the increased time cost. This is why processing different files requires slightly different amounts of time.

We have also investigated the filtering of data using a few sequential additive filters, in order to derive simplified filters with fewer computations. Presorting operand arrays can significantly speed up the filtering.

We have evaluated the volume of input data for a run unit in order to integrate a fast data interface between the operational input data source and the new PC platform via Internet. The total volume of inputs for a unit run is 23.60 MB. Only the ir and vis files are dynamic files of combined volume 12.838 MB or approximately 130 Mbps and can be ingested into the PC platform using a 100 Mbps Ethernet interface in acceptable time of a few seconds. This was used to design the new operational ground system configuration to include the heritage and conversion platforms in a local point-to-point area network, as described in Section 8.

With the completion of these basic investigations of IDL specifics and the IDL application performance, we were in a better position to conduct timing tests, find the bottlenecks, recover the new platform bottleneck algorithm's definition from the IDL code or published sources pointed to by the IDL reference manual. From this point we proceeded to make a few obvious code improvements and optimizing the algorithms in the IDL code, mainly by using different command options and processing array data as vectors were applicable. The timing results and the efforts of algorithm's definition recovery and optimization are described in the following section and the rest of the paper.

4.1 IDL Data Structure Specifics

Similar to the basic argument type of Matlab being a matrix, the basic argument type in IDL is an array. It is important to clarify the IDL definition of an array dimension and size. The dimension definition is the classical one – the number of subscripts needed to reference an array element. For example, array A(3,3) has two dimensions. This is exactly the same as a matrix dimension definition in Matlab. Each dimension is numbered from left to right as dimension 1, dimension 2, etc. In Matlab and C, dimension 1 corresponds to a row and dimension 2 to a column. It is the opposite in IDL. To correctly process an array that is passed as a parameter from IDL to a C-function, indices in C must be in reverse order compared to that in IDL. Size is pertinent to a dimension and size is the number of subscript values that is needed to reference all elements in the array along this dimension, beginning with subscript 0. IDL has internal limitations on array sizes, which are different from the limits imposed by the platform operating system. For example it does not allow allocation of a byte array of size 1 Giga Byte (GB) while Windows operating system allows addressing of 2 GB of memory.

When a new or an existing array is equated to an expression, it preempts the type of the longest type argument in the expression. However, partial array assignments leave the type of the resulting array unchanged. This is natural since in a partial assignment it is unknown how the rest of the array is going to be assigned, and IDL prudently leaves the original array type unchanged and instead implicitly casts the type of the argument arrays into the type of the initial array. This is an important point because when the assignment expressions contain just array names and at least one DOUBLE array operand, the resulting array type becomes DOUBLE, and thus requires twice more memory. For example, one of the three resulting arrays recomputed in the bottleneck code segment are originally of single precision floating point type and each occupying 43MB of RAM. After a recomputation that involves an argument of double precision floating-point type the array type is changed to DOUBLE and requires twice more RAM. This may not be necessary for the result precision and can be avoided when code is implemented in hardware. Whenever such a saving is planned at the apparent expense of accuracy, the science algorithms originator must approve it. We introduced partial array processing and array interpretation as vectors for pre=hardware optimization tests and for RC implementation.

4.2 User Implemented Loops

There are advantages and disadvantages in implementing user-constructed loops. IDL functions for operations on arrays provide better time performance than for-loops implementation by users for small arrays when all computation arguments fit into the RAM heap. However, for large arrays, as in the GOES-8 application this is not the case. When the PC platform does not have sufficient internal memory, it reverts to virtual memory on the disk for IDL code segments and large dynamic memory allocation. For example, the GOES-8 application required 20 minutes for the basic operation of producing one large output color picture with 128 MB RAM memory and 12 minutes with 256 MB memory. Replacing some IDL computations over

arrays by partitioning reduces the need for one-time-memory and the number of disk accesses, and reduces computation time to less than 7 minutes. Of course, the array operations under consideration are element-by-element operations, allowing such direct array partitioning. We have performed time cost analysis (Table 1 Section 4.3) of one large loop and a few nested loops without any operations in the loop implemented in IDL. This time costs are higher than expected for large loops (10 seconds for a 3600x3000 size) leading to the lesson learned that these loops in IDL procedures that use large loops and have performance under 10 seconds were implemented in IDL system in some other compilation language with its executable being embedded in the IDL system.

4.3 Elementary IDL Operations Cost

Different experiments were conducted to determine the time cost of a few basic IDL operations, like +, *, <, >, assignment and indexing. This was done in order to determine segments of code where improvements would be guaranteed. There are three modes of processing: nominal (small), medium and large. The cost of an operation was determined by timing an IDL FOR loop of length 1800x1500 (problem case 'medium') and having in the loop a different expression with basic operations. A medium size loop with external call clocked the same time as a loop with an IDL call. Following is Table 1 summarizing operation costs. Table 1 was obtained by timing a medium size loop (2.7e6 loop counts) and a 128MB PC RAM configuration.

Table 1. Time cost of IDL operations

IDL Operation	Time Cost (sec)	Expression Timed	FOR Loop Size
+	2.5	a=1.0 + 2.0	2.7e6
*	2.5	a=1.0 * 2.0	2.7e6
+ and subscripts	4.29	v=f[i] + f[i]	2.7e6
subscript	1.7		2.7e6
++	4.51	a=1.0+2.0; a=3.0+4.0	2.7e6
<	3.13	v=v < 0.0	2.7e6
=	2.57	v=1.0	2.7e6
++	10.0	a=1.0 + 2.0; a=3.0+4.0	1.8e7
no operation	1.3		1.8e7

5.0 Timing Tests on the New Platform

Timing test runs were conducted on the PC/Windows98 platform in order to collect timing results before algorithms and code optimization and hardware implementation. The IDL source algorithms and code performance timing results on the PC platform were then analyzed in order to determine the bottleneck code segments for hardware implementation. The application time complexity was evaluated on the PC/Windows98 platform for the three required configurations of problem size – small medium and large, and variable PC RAM memory size. The results of these tests are described below in Table 2.

5.1 Timing Results

The real-time processing performance criteria "**RT=75 seconds**" was defined above in the Introduction. The GOES-8 development and operations team provided the two UNIX native platforms' Benchmarks UB1, UB2.

Table 2. GOES-8 Application Pre-Hardware Timing

Platform	Image Columns	Image Rows	RAM (MB)	Time (sec)
PC	900	750	128	16.00
	900	750	256	16.00
	900	750	768	14.00 (RT/4)
UB1, UB2	900	750		58.45 21.60
PC	1800	1500	128	80.21
	1800	1500	256	36.00
	1800	1500	768	32.00 (RT/2)
UB1, UB2	1800	1500		90.00 39.54
PC	3600	3000	128	1200.00
	3600	3000	256	760.00
	3600	3000	512	150.00
	3600	3000	768	125.00 (RT+50)
UB1, UB2	3600	3000		291.00 106.00

5.2 Development System Cost

The cost of the PC system was $2533. The cost of the IDL development system for the PC was under $2K and the cost of the memory upgrade was $1K. The cost of MS Visual C++ 6.0 is $800. The cost of the FPGA/DSP board system is under $2K.

6.0 Optimization and RC Implementation

The software application's bottleneck algorithms were determined, optimized and implemented in RC hardware and software.

The issue of an acceptable definition for a computational algorithm is still open. The work in this area is continuing in the USA and international bodies that are developing related standards. When this issue is encountered in software engineering, the algorithm definition in many cases requires reconstruction, refinement, and optimization.

For the benefit of the GOES-8 application conversion project, we give an operational definition of an algorithm. We hope that this definition will be useful and practical for other projects' software and hardware implementation by personnel versed in software and hardware programming. An algorithm definition must be susceptible to direct and rapid software or hardware implementation. All implementations should produce the same required or better output accuracy for identical inputs. The algorithm definition, provided by documentation and the code execution examples must be repeatable with manually computed results. In case of discrepancies, there is a need to go back to the code requirements or even one more step back to the Algorithm Theoretical Basis Definition (ATBD) provided by an instrument science team. The ATBD, in our view, is more "a to be further defined" phase for an algorithm operational definition that is useful for engineering implementation.

6.1 Pre-Hardware Optimization

Porting the application from its native platform to the PC platform, a few elementary enhancements to its IDL code, upgrading the development PC platform to its maximum memory capacity of 768 MB, and introducing the largest bottleneck process split into 16 segments, allowed achieving the 14 seconds performance improvement (a factor of 4) for generation of the small size picture, and 32 seconds were required for the medium size picture generation, improvement by a factor of 2.5. This performance was reached even before the code implementation in hardware. All extraneous processes on the platform were terminated before the IDL application run. Not initializing a large array to zeros when all the elements are computed later or allocating resource just before they are needed resulted in improved performance. Also, in one case, a large array computation expression was simplified by replacing a computational component with a constant. A single term was introduced to replace its multiple computations in the largest bottleneck code segment. A unique splitting of a large array handling process into sixteen segments reduced concurrent requirement for heap memory by a factor of 16 and improved performance for the large color picture generation. However, the 120 seconds required for the large size picture generation was difficult to improve on the PC platform, even as it is already at (RT-50) seconds. It is also difficult to further improve the application performance for the small and medium size images to allow higher telemetry rates because the IDL procedures used in this application are already optimized. Whilst the need for the GOES-8 application bottleneck code hardware implementation becomes extremely evident and necessary, the approach taken for this application is to implement in hardware only the bottleneck code segments. We are also taking advantage of the application data processing parallelism nature that was established above in its application model and propose to use two or more RC components for implementation of independent bottlenecks once a solution for one bottleneck is achieved.

6.2 Application Bottleneck Algorithms

The algorithms of the few critical IDL functions and procedures we selected for possible optimization were reconstructed by experimentation and reaching a consensus with IDL vendor on their IDL implementation. The following algorithms were analyzed in detail, optimized in IDL or C++, and the shifted bottleneck code segment was the first to be implemented in RC hardware. It is important that a selected function local optimization does not degrade the entire application system performance.

6.2.1 Interpolation

The application is using the IDL bilinear interpolation procedure INTERPOLATE for which the algorithm or source code are not available. The INTERPOLATE is used *once* in the application to derive angles of the sun altitude on a 3600x3000 grid using pre-computed angles on a 100 times smaller 361x301 grid. It was requested by the GOES-8 project that this procedure time performance be improved. We reconstructed the IDL interpolation algorithm based on run time analysis of the procedure and then optimized the algorithm using the application specific a priory knowledge about indices in rectangular arrays and similar to [3].

6.2.2 Fast Fourier Transform

The application is employing in one user procedure the Fast Fourier Transform (fft2) on a two-dimensional (2-D) ir4 band image. The fft2 is used to *correct and sharpen* images obtained by spacecraft in r4 band. The direct fft2 is performed on ir4 file image and the lower frequencies are then filtered and smoothed after which the backward fft2 is performed. The result is superimposed on the source image, sharpening the ir4 image. It was requested by the GOES-8 project that this procedure time performance also be improved. The work towards satisfying this request was based on the observation that the sum of prime factors of source image sizes is equal to that of array sizes closest approximations by numbers of power two. The fft computational complexity is known to be proportional to this sum. Performing fft on the original arrays saves RAM by a factor of 1.5 and contributes to better overall performance. There are also very fast fft cores for FPGA and DSP boards.

6.2.3 Color Systems Conversion

The application is using two color systems to describe images namely, the {r,b,g} and the {h,s,v} systems. The necessity to convert from one to another is probably dictated by the necessity to apply to images digital 0 and 1 cutoff filters. The {r,b,g} to {h,s,v} conversion maps the {r,b,g} color values into range 0 <= {h,s,v} <= 1 that facilitates this filtering in IDL, using IDL operations of maximum (>) and minimum (<) on an image matrix and corresponding constant of 1 or 0. After filtering the image is converted back to {r,b,g} that is a convenient color system for image visualization and display. IDL provides the necessary procedure COLOR_CONVERT with an option

RGB-HSV and HSV-RGB similar to [4]. However, this procedure is notoriously slow. The IDL color conversion functions performance also depends on input data content, as much as by a factor of 2 for large images. It was requested by the GOES-8 project that this procedure time performance also be improved. The IDL algorithm was reconstructed and implemented in C++. This resulted in code better performance and could facilitate further performance improvement by implementing the C-code on a DSP board.

6.3 Digital Filters

Multiple digital 0-1 filters are constructed and used throughout the application. These filters are applied to large arrays on a pixel basis and can be viewed as vector operations. The time complexity is driven by vector size, data content, and the number of filters applied. The filters are implemented using the IDL maximum and minimum functions (>, <) and some threshold constraints. These filters comprise one of the bottlenecks and could also be implemented in RC hardware.

6.4 Bottleneck Code Segment Shift

It can be observed from Table 3 that the benchmark timing performance bottlenecks shifted from "interpolate", "sharpen ir4", "calc v,s,b", "calc v,s,b with vis" and "color_convert" combined cost of 35% to 20% on the new platform.

Table 3. Platform Performance Focus Shift

IDL Function	UB1S	PCS	UB1L	PCL
ir,vis,pseudo -intp	3.22	1.00	30.10	4.00
sza **(interp)**	0.57	2.09	**10.27**	2.09
sharpen ir4 **(fft2)**	27.05	5.05	**26.76**	4.83
calc v,s,b	1.66	0.76	**27.50**	**11.87**
bump size	0.44	0.44	15.00	6.15
read vis	11.00	8.00	11.00	3.52
calc v,s,b w/ vis (shifted bottleneck)	4.27	1.70	**73.71**	**70.0**
color convert	1.43	0.55	**22.85**	**19.00**
Unit files time	59.00	14.00	**291.00**	**125.00**

It appears that implementing the "calc v,s,b with vis" code segment in RC hardware may gain the time needed to meet the real-time performance criteria of 75 seconds. This is the rationale for focusing the hardware implementation on this segment of code first. This is also the rationale because it is susceptible to problem partitioning and parallel implementation as can be seen from the application model. Note that memory increase shortened large input vis file read because its buffer is now completely in RAM memory and not partially in virtual memory on the hard drive. Similarly, implementing "calc v,s,b with vis" in hardware, will free PC RAM and improve the performance of the following software function color convert. Table 3 summarizes the timing for cases UB1 Small (UB1S), PC Small (PCS), UB1 Large (UB1L), PC Large (PCL). All PC timing was obtained using the 768 MB upgraded memory.

Note, that pseudo-intps processes are not using interpolation. The application running on the PC platform required for smallest image 052MB of heap memory and for the largest image case 549MB of heap memory. Since the PC memory was upgraded to 768MB most of the processing requires less disk IO and explaining the interpolation, fft2 and color convert procedures performance improvement. The goal now is to implement the bottleneck "calc v,s,b with vis" IDL code in FPGA or DSP hardware to achieve the real time benchmark performance of 75 seconds from the present 120 seconds.

6.5 Baseline RC Library

The accumulation of RC implemented functions and tools to be used in a PC/WINDOWS/RC environment is incremental and will evolve over a period of time. The development of tools for debugging RC hardware is of most importance. In September-October of 2000, students from WPI each implemented a Matlab function, for example the simplified linear interpolation, statistical mean, and standard deviation function in RC hardware. This was the initial effort to evaluate the complexity of such an effort with available FPGA and DSP boards and resulted in the development of an FPGA floating point functions of multiplication, addition and comparator. The proposed future work plan is to develop a baseline library of Matlab, IDL, C and Java (Trademark of Sun Microsystems Inc.) functions implemented in RC. The Annapolis Micro Systems released its floating-point arithmetic library in spring 2001. This will significantly facilitate the development of the Baseline RC Library.

7.0 SOLUTION DISCUSSION

7.1 PC/Windows/RC Platform Configurations

The original PC/Windows98 platform was a DELL Dimension XPS T550, with a Pentium III 550 MHz processor, 128 MB RAM and a 20 GB hard drive. Its memory was upgraded to 768 MB via the replacement and additions of 2 256 MB PC 100 SDRAM Non-registered modules. To overcome the PC BIOS memory limitations, the BIOS was upgraded from Version A04 to A09, obtained from DELL. Also added to the PC was the TMS3206701 DSP EVM (evaluation model board) board from Texas Instruments, Inc. The DSP carries 2 external memory banks, 4 MB each. The bottleneck code was implemented as the DSP target module.

An alternative delivery PC platform is based on a DELL XPS B933 Series Pentium IV Processor of 933 MHz, 1GB RDRAM memory at 133 MHz and a 75GB Hard Drive. The system is running the Windows NT or Windows 2000 Operating System (Trademark of Microsoft Corporation). The PC also carries the FPGA StarFire board from Annapolis Micro Systems Inc. This device software driver is only officially available for Intel x86 PC based platforms running the Windows NT operating system. The FPGA board has been programmed in VHDL to execute the GOES-8 application bottleneck code segment and the same bottleneck code was implemented in the DSP target module.

The series of a few large floating-point value vectors, that are operands in the bottleneck computational expressions, are passed using IDL EXTERNAL_CALL to the PC host module. This host module was developed in Microsoft Visual Studio C++ Version 6.0. This Language environment is compatible with the FPGA/DSP host functions library and the board PC driver for Windows 98, Windows NT, or Windows 2000 Operating Systems developed by the RC board vendors. The host module accepts the large vectors' pointers and transmits the data to the FPGA or DSP board for intensive computations. The host program also obtains the resulting vectors from the FPGA or DSP board into their original input vector place using the IDL pointers. When the host module is finished, it returns control to the IDL code. The IDL application then continues till completion. The host source code is compiled and linked to create a required by IDL DLL file that is placed into the IDL primary directory. The IDL/TI vendor "make files" were used to generate this DLL file. Extensive documentation was developed to describe the development procedures for both host and target module development, integration and testing. A more powerful PC may be useful for speeding up the FPGA code reconfiguration. A powerful PC platform may even eliminate the need for RC components, however the goal is to achieve real time performance with minimum resources as to make the system more feasible for a future space flight opportunity.

7.2 RC DSP Solution Specifics

These GOES-8 application bottlenecks are good candidates for their code implementation in RC hardware that is specialized to perform well in integer data processing-intensive applications. However, the prevailing general point of view was that data processing, involving floating point operations, is not susceptible to direct RC hardware implementation. The alternative experimental solution involved a very costly conversion of floating point type data to integer types and implementation of an entire software application in hardware. We have implemented the GOES-8 application bottlenecks code using floating-point arithmetic on the FPGA and DSP boards.

The multi band data source specific requires partition of the PC-RC data transfer stream into telemetry band segments or aggregated data segments. This specific necessitates a need to determine two parameters **n**, **k**. The first is the number of columns from a data band array that is used in one transfer and the second is the number of data transfer blocks for a unit of operations. A transfer comprises n-columns from each data source in separate IOs or one IO of aggregated data from all sources. The two parameters n, k are bounded by a few Diophantine equations derived from the fact that an image is a 2-dimensional array of size (**C**, **R**), where **C** is number of columns and **R** is the number of rows in IDL notation or number of pixels in a column. Furthermore the RC hardware boards carry a limited amount of external memory banks (DSP SDRAM) each of size **M**. Let g be the number of data source arrays and Sj a pixel size in bytes, where 1<=j<=g. The relations that describe the n, k are:

n, k are integers and k<=n.
n x k = C (due to array structure)
n x R x (S1 + S2 + ... + Sg) <= M

The GOES-8 project solution for DSP is **n=k=60.**
This is determined by the corresponding parameters

C=3600, R=3000, M=4x10**6,
S=S1+S2+S3+S4+S5=8+4+4+4=20.

Finding the largest value **n** delivers the fastest data transfer time. Furthermore, within each of the **k** transmissions of **n**-columns of all multi source arrays' data, different prearranging schemes were tested. This was necessary in order to facilitate the RC component fast operands access from its external memories. Initially **g** IOs in sequence (each **n**-columns long) were performed on the PC for each of the **g** data sources and in a loop of size **k**. The data destination was the presently available external memory bank on the DSP, beginning at memory entry address and utilizing SxnxR or 3.6MB of the 4MB memory bank. The second prearrangement was to collect all operands, that are used by the DSP module in one pass of its main computational loop, into one consecutive group (quadruplets) within a single transmission buffer on the PC. However, this required that all inputs have the same data type and input described above by S1=8 was casted in IDL from double to single precision floating point type and S1=4. The source data buffer length for this prearrangement was 2.88MB. Preserving the **k**-size IO loop and pre-arranging in the PC host module a single transfer buffer of quadruplets of pixels from all the **g** data sources before buffer transmission to the RC board alleviated the multi source data access problem on the DSP and the PC-DSP IO interface bandwidth problem.

A few words are due on the concept of pipelining. There is work to be done before pipelining becomes possible – pipeline route survey, preparing the route, pre-processing input data streams and after-pipelined computations laying out the pipelined results in sequence. The literature concept implies that this work is already done and is negligible. This is not the reality in image data processing, where it is at the

core of the overall performance problem, especially for an FPGA board.

Selection of PC/DSP data transfer mechanism must ensure that that the computational power of the RC board is not offset by the best data transfer rate limitations. RC board vendor performance claims must be verified under the same conditions that were used by the vendor in order for the claimed benchmarks to have any meaning. The GOES-8 project fastest data transfer mechanism was determined to be the Host to DSP Peripheral Interface Port (Hpi) high-level host library functions.

There is a need to introduce a specific and fast PC-RC operations API synchronization mechanism. Although the choices are many, finding the acceptable one is a subtle problem. For the GOES-8 project and DSP we selected two system memory semaphores on the DSP board that are set and monitored by the PC and DSP. There are specifics in handling a wait state on a semaphore. While all DSPs are equipped with a multiplier-adder-store or multiplier-accumulator path that can produce a result in one instruction cycle, the internal pipelining may result in latency more than one cycle. Only when application code constitutes a long series of multiple-accumulate operations (a so-called DSP application) can DSP achieve its advertised performance. If a multiply operation is preceded and followed by other kind of operation, more than one instruction cycles are spent waiting for the multiplier result. The application code must be thus re-structured into a DSP application for performance problem solution using a DSP.

7.2.1 RC/DSP Implementation Outline

The implementation of the bottleneck code segment on the DSP board followed the same methodology as was used in the project conversion. The roadmap to this methodology is first to convert the bottleneck code from the PC IDL code to the PC/Host C++ and DSP/Target code. Then make it work on this PC/RC platform to produce results comparable to the PC/software only generated color pictures for the same test set of input files. The workable PC/RC code application is then optimized to achieve comparable to the PC platform IDL software code or better run-time performance. And finally, optimize the RC target module to achieve real-time performance.

The RC/DSP implementation consists of introducing a new case value 'large_dsp' for the problem size switch. The main processing IDL procedure is then using this switch to activate the RC implementation mechanism.

```
case switch_arr_size of
  'large_dsp':
    begin
v_float=TEMPORARY(v_float)/(this_s_alt+0.1)
f=float(f) ;To make all parameters of same type!
CALL_EXTERNAL(lib_name('dmahost8'),
'dmahost8', f, v_float, new_v, new_s)
new_b=f*ir_float1*225+(fff)*temporary(new_b)
    end
endcase
```

The RC/DSP implementation is comprised of the PC host and DSP target modules and associated make files - dmahost8.c, dmahost8.def, dmahost8.mak dmatarg8.c, link1.cmd, dmatarg8.mak, and dmastates8.doc. The last is a documentation file that contains the host and target module synchronization states diagram. When the two make files are executed from an MS-DOS window two files dmahost8.dll, dmatarg8.out are created. This completes the GOES-8.prj project RC/DSP configuration and the project is ready for run to generate the large color picture using the RC/DSP implementation. It takes a few minutes from a software change, for which the results are assured, to the configuration of the host and target modules and project execution.

7.2.2 PC/DSP Board Data Transfer and CPU Operations

The DSP memories' structure allows concurrent access to each type of memory by the PC/DSP Processors or PC/DSP DMA Controllers. For example, the TI SDRAM is comprised of two banks, Bank0 and Bank1 or for short - B0 and B1. This allows the application design topology where host data transfer to and from the DSP board is concurrent with DSP computations of the bottleneck expressions. This depicts the possibility of the DSP part of the application time complexity being closely approximated by

$$\text{DSP(Time)} = \text{Time(IO/DMA)} + \text{Time(DSP Processors)} = \text{Time(IO/DMA)} + \text{Deltat},$$

where Deltat is small and can be ignored. This then makes the DSP performance acceptable even at slower than expected data transfer rates between the PC and DSP over the PCI bus.

7.3 RC/FPGA Implementation Outline

The implementation of the bottleneck code segment on the FPGA board involved many steps in order to ensure calculated results did not vary from original software versions of the algorithm. Initially, floating point mathematical functions were developed and implemented on the FPGA. Once results from these functions were verified, they were streamed as required by the algorithm to produce the interim final image. The FPGA board algorithm was then transformed into a function call from IDL so that the GOES-8 application could be run seamlessly from end-to-end, with the FPGA board interface being transparent to the user.

Initial development of the FPGA board algorithm segment (henceforth referred to as the FPGA algorithm) was not trivial. Interfaces between the FPGA and on-board memory and floating point mathematical functions including addition, subtraction, multiplication and comparison were developed. The commercial of the shelf (COTS) RC board

used for this development was the Annapolis Micro Systems StarFire board. The StarFire board used utilizes a Virtex XCV400 FPGA and has 1 MB of on-board SRAM, which is divided into 2 banks of 500 Kbytes each, commonly referred to as the left and right banks. The initial run of the FPGA algorithm utilized only the left bank of memory and processed one set of data inputs at a time. Although this method completely underutilizes the performance of the FPGA, it was important to run the algorithm in this mode for debugging purposes and in an effort to validate the output data.

Once the output data was validated, efforts to optimize the FPGA algorithm began. The first optimization step was to run the algorithm using both banks of memory. This was accomplished using a "ping-pong" method whereby one bank of memory gets loaded by the processor over the PCI bus while the other bank of memory is processed in the FPGA. Implementation of "ping-ponging" memory cut execution time of the FPGA algorithm by a factor of 2. At this point in time, a COTS floating point math library became available which utilized full pipelining with valid-bits. This library contained all the math functions used by the FPGA algorithm with the exception of a floating-point compare. The COTS floating point math cores were benchmarked against our previously developed math functions resulting in decreased execution time of the FPGA algorithm. This led to the development of a pipelined comparator with the same input and output characteristics as the COTS math cores.

7.4 PC/RC/Windows Platform Performance Timing Results, Comparison and Analysis

We have reached and exceeded the real-time performance for all required cases of the project, namely 14 seconds for small, 32 seconds for medium and 75 seconds for large color picture production as compared to the real time requirement of 75 seconds. We have solved the GOES-8 application color picture product generation in real time problem using a 550MHz PC running Windows98 Operating System and a TMS320C6701 Evaluation DSP board. This platform is processing all five bands 2MHz telemetry source in real time. The problem was also solved using and FPGA board. Another experimental project using an FPGA board allows real time data processing of 5 bands from the Terra/MODIS spacecraft 36-band 10MHz data source or a 1.4MHz data stream. The GOES-8 project solution demonstrates that the MODIS application could be similarly implemented using the proposed methodology and RC hardware floating point libraries.

7.4.1 PC-DSP IO Timing Results

The best data rate achieved using the DSP board asynchronous data transfer mechanism was 12.8MBs. The achieved data rate using the PC/DSP Host Peripheral Interface (HPI) mechanism is 10.4MBs. However, it allowed DSP to compute the bottleneck code concurrently with data transfer, and the total time is comparable to the data transfer time, or 25 seconds. The HPI mechanism was selected for the application data transfer between the PC and the DSP.

7.5 Reconfiguration Time

A DSP reconfiguration for a small code change is comprising recompilation of host and target modules, integration and rapid testing. The DSP target module compilation and load onto the DSP is rapid, allowing fast system reconfiguration.

8.0 New System Configuration

The new system configuration (Figure 2) comprises portions of the existing native SGI/Workstation/UNIX platform and the new PC/RC platform connected by the Internet.

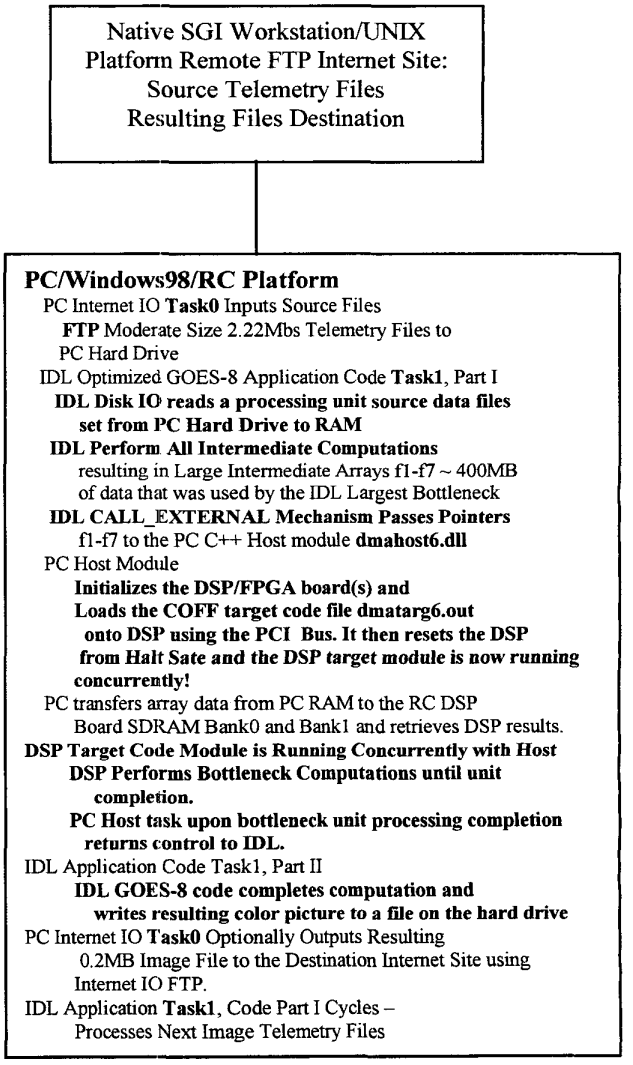

Figure 2. Solution Real Time Processing System

The new system makes use of portions of native processing, that are preceding the GOES-8 native application and are

maintaining the input files. An Internet task is running on the new PC/RC platform and constantly seeks out on the heritage platform a set of input files for the next image. It ingests it onto the new platform using the common Internet file transfer protocol utility (ftp). The new PC/RC GOES-8 application generates the color pictures in real-time and ftps them back to the native platform. This configuration can be expended in the future to use a few FPGA or DSP boards for next highest priority bottlenecks implementations in hardware. These secondary bottleneck segments again become prominent, although on a lower level, after the main bottleneck code segment is implemented in hardware.

CONCLUSIONS

We have accomplished the goal we set out in the introduction and achieved real-time performance of the GOES-8 color picture generation application. We have developed a new ground system configuration for this specific project and laid the basis for its re-use in future projects, including spacecraft on-board data processing. We have developed a methodology that widens the field of telemetry processing for RC technology applications that require large intermediate multi source data IO and concurrent intensive computations. This result also allows support for other GOES products applications. We have learned some lessons, including the lesson that cooperation of the application science originators/users, application developers and development systems' vendors is crucial to such a project success.

REFERENCES

[1]. T. Flatley, "Enabling the Earth Science Vision through Reconfigurable Computing",
AIST-0132-0000, May 1999

[2]. M. Figueiredo, P. Stakem, T. Flatley, T. Hines, "Extending NASA's Data Processing to Spacecraft", IEEE Computer Magazine, pp. 115-118, June 1999

[3]. S. Kizhner, "On Fast Post-Processing of Global Positioning System Simulator Truth Data and Receiver Measurements and Solutions Data", Proceedings of ION GPS'2000, 14-17 September 2000, Salt Lake City, Utah

[4]. James D. Foley, el al
Computer Graphics Principles and Practice,
Addison-Wesley Publishing Company, 1990

[5]. PCI Local Bus Specification Revision 2.1 by PCI Special Interest Group, 1995

BIOGRAPHY

Semion Kizhner, an aerospace engineer with the National Aeronautics and Space Administration (NASA) at the Goddard Space Flight Center (GSFC), participated in the development of the Space Shuttle launched Hitchhiker carrier and several attached Shuttle payloads such as the Robot Operated Materials Processing System (ROMPS). He was responsible for establishing the Global Positioning System (GPS) test facility at GSFC and supported GPS simulations for several space projects, such as the OrbView-2, SAC-A and EO-1 spacecrafts [3]. He is currently developing capabilities to access spacecrafts as nodes on the Internet and to accelerate generation of images derived from the EOS Terra spacecraft MODIS instrument and GOES-8 spacecraft data. He graduated from Johns Hopkins University with an MS degree in computer science.

David Petrick, an electrical engineering student with the GSFC Electrical Systems Center Ground Systems Hardware Branch, is currently working on implementing the GOES-8 image processing bottleneck software into reconfigurable hardware. He is studying at the University of Pittsburgh with a concentration in the area of signal processing and telecommunications.

Tom Flatley is an electronics engineer at the NASA/GSFC and is currently the Associate Head of the Ground Systems Hardware Branch. From 1993 to 1997 he had served as the head of the Flight Electrical Systems Section and Flight Component Development Group. Prior to 1993 he developed numerous flight and ground components and subsystems for various NASA missions. Mr. Flatley's current work includes the coordination of Reconfigurable Computing (RC) development activities for the GSFC Electrical Systems Center, with emphasis on developing RC technology for flight applications. This includes the management of work in flight component, ground-based functional prototype and development tool areas in collaboration with government, industry, academic partners, and teamed with applications from end users in the science community [1], [2].

Phyllis Hestnes is an electrical engineer at NASA Goddard Space Flight Center. She graduated from the University of Maryland. Her primary interests include the design of processor based systems and reconfigurable computing.

Marit Jentoft-Nilsen is a computer engineer at the NASA/GSFC/SSAI. She graduated from the California Institute of Technology and her primary interests are in satellite image data processing. She has been developing the GOES satellite telemetry data processing applications since 1993.

Karin Blank is a computer engineer with NASA, Goddard Space Flight Center, and is currently working on implementing GOES-8 algorithms on a digital signal processor. She recently graduated from Worcester Polytechnic Institute with a BS degree in computer science.

ACKNOWLEDGEMENTS

The authors would like to thank Dr. Dennis Chesters/NASA GSFC for support at all stages of this project. We also would like to thank the Worcester Polytechnic Institute students and their faculty advisors for their readiness to tackle difficult short-term assignments in support of this project – Keith Leveille, Jay Bose, Kenda Conklin, John Hammond, Mark Hados, Karin Blank and professors Fred J.

Looft, Donald N. Zwiep, David C. Brown. We are also indebted to a large number of people from industry, who have helped us with issues of their company products - from Texas Instruments Navaid Karimi, Larry Stapleton, Rekha Ramadas, Kevin Jones for help with DSP board issues; from Research Systems Inc. Doug Loucks, James Jay Jones, Adam Bielecki, Atle Borsholm, Carol Vandenbelt, Valiant Villanueva for their excellent technical support.