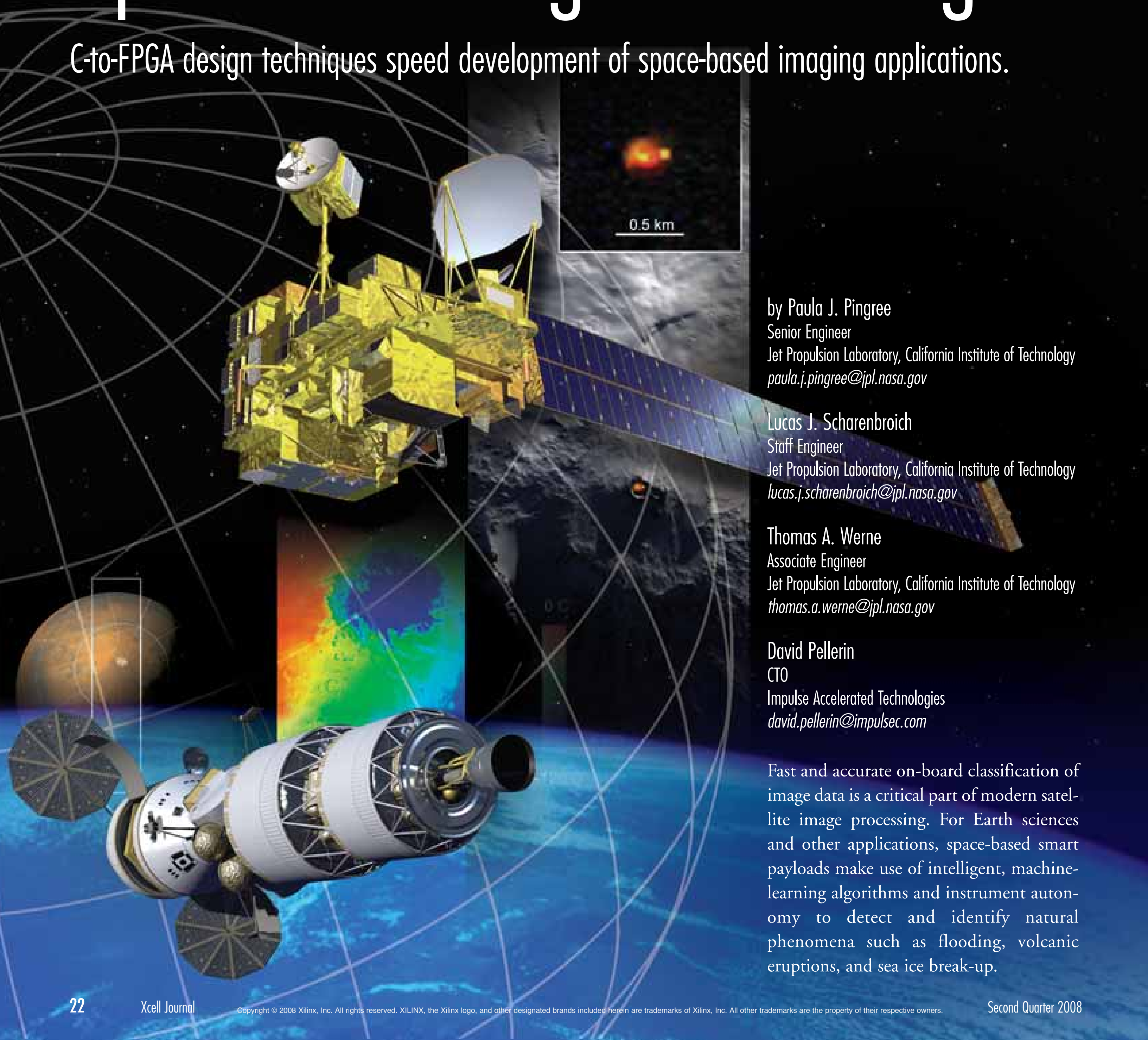
Developing FPGA Coprocessors for Performance-Accelerated Spacecraft Image Processing

C-to-FPGA design techniques speed development of space-based imaging applications.



by Paula J. Pingree
Senior Engineer
Jet Propulsion Laboratory, California Institute of Technology
paula.j.pingree@jpl.nasa.gov

Lucas J. Scharenbroich
Staff Engineer
Jet Propulsion Laboratory, California Institute of Technology
lucas.j.scharenbroich@jpl.nasa.gov

Thomas A. Werne
Associate Engineer
Jet Propulsion Laboratory, California Institute of Technology
thomas.a.werne@jpl.nasa.gov

David Pellerin
CTO
Impulse Accelerated Technologies
david.pellerin@impulsec.com

Fast and accurate on-board classification of image data is a critical part of modern satellite image processing. For Earth sciences and other applications, space-based smart payloads make use of intelligent, machine-learning algorithms and instrument autonomy to detect and identify natural phenomena such as flooding, volcanic eruptions, and sea ice break-up.

The Jet Propulsion Laboratory (JPL), a National Aeronautics and Space Administration (NASA) laboratory, has developed support vector machine (SVM) classification algorithms used on board spacecrafts to identify high-priority image data for downlinking to Earth. These algorithms also provide onboard data analysis to enable rapid reaction to dynamic events (Figure 1). These onboard classifiers help reduce the amount of data downloaded to Earth, greatly increasing the science return of the instrument.

SVM classification algorithms are flying today, using computational platforms such as the RAD6000 and Mongoose V processors. These legacy processors have only limited computing power, extremely limited active storage capabilities, and are no longer considered state-of-the-art. For this reason, onboard classification has been limited to only the simplest functions running on only a subset of the full instrument data: for example, only 11 of 242 bands in the case of the Hyperion instrument on the Earth Observing-1 (EO-1) satellite.

FPGA coprocessors are an ideal candidate for these algorithms. FPGAs can provide significant improvement in onboard classification capability and accuracy when compared to the legacy processing platforms now flying.

To evaluate the effectiveness of FPGAs for SVM algorithms, we implemented a legacy snow-water-ice-land (SWIL) classifier, originally developed for the Hyperion instrument, on the Xilinx® Virtex™-4 FX60 FPGA. To develop the application more quickly, we took advantage of the Impulse C-to-FPGA compiler tools provided by Impulse Accelerated Technologies. These tools support the rapid development of highly parallel hardware algorithms and applications.

This article describes our approach to implementing the Hyperion linear SVM on the Virtex-4 FX60 FPGA, as well as additional experiments that we performed using an increased number of data bands and a more sophisticated SVM kernel. These experiments show the potential for more efficient, higher performance onboard classification using FPGA-embedded algorithms.

FPGAs for Onboard Computation

Onboard computation has become a significant bottleneck for advanced, space-based scientific and engineering applications. Currently available spacecraft processors have high power consumption, are expensive, require additional interface boards, and are limited in their computational capabilities.

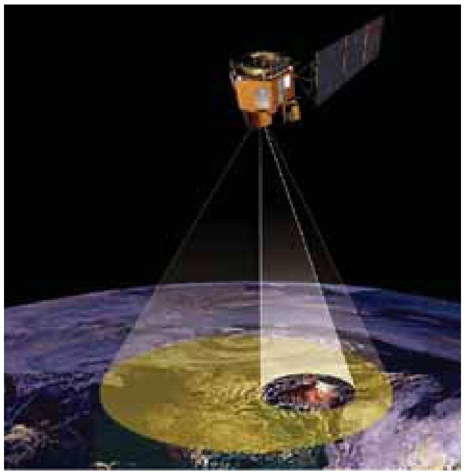


Figure 1 – NASA uses smart payloads to classify Earth image data and reduce the amount of data required for downloading (Image courtesy of NASA).

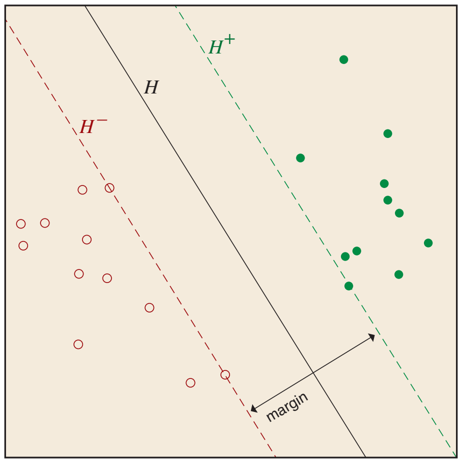


Figure 2 – Calculating a separating hyperplane using an SVM. The circled data points are the support vectors that lie on the margin.

Recently developed hybrid FPGAs, such as the Xilinx Virtex-4 FX device, offer the versatility of running diverse software applications on embedded processors while at the same time taking advantage of reconfigurable hardware resources, all on a single chip. These tightly coupled, single-chip hardware/software systems offer lower power and lower cost than general-purpose, single-board computers (SBCs). FPGA platforms offer breakthrough performance over radiation-hardened SBCs, leading to entirely new architectures for smart payloads.

To evaluate the potential for acceleration using FPGAs, we selected the Xilinx ML410 evaluation platform for development and demonstration of selected smart payload concepts. The Xilinx ML410 evaluation board comes equipped with a Virtex-4 FX60 FPGA that features two embedded PowerPC 405 processors and a large amount of available FPGA logic.

The specific algorithm we chose for our investigations is the SVM classification algorithm. Algorithms of this type have found broad application in general machine learning and classification tasks, as well as for onboard remote sensing. An SVM is a maximum margin classifier that calculates a separating hyperplane between two labeled classes such that the distance to the nearest data in each class is maximized (Figure 2). By selecting such a maximum margin hyperplane, the SVM classifier can exhibit better generalization to new data than other linear classification methods.

The goal of training a support vector machine is to learn a set of weights such that the sign of a weighted sum of dot products between the training data, x_i, and a test vector – t – will correctly predict the class of the new data vector.

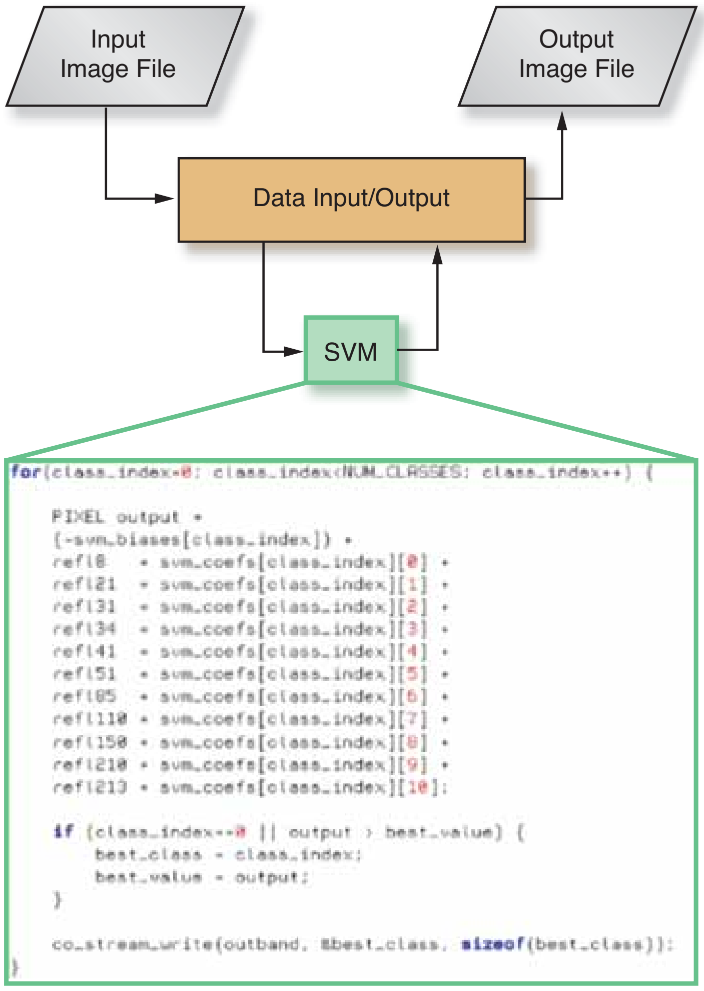
SVMs also incorporate what is known as the kernel trick, a method allowing them to be extended from purely linear to non-linear classifiers. This method involves formulating the training and testing algorithms in terms of dot products and then replacing the dot products with a kernel function that represents a dot product after passing the arguments through some non-linear function. The kernel function permits the high-, or even infinite-dimensional, dot products in the non-linear space to be computed using terms from the original, low-dimensional space.

SVMs are well suited to onboard autonomy applications. The property that makes SVMs particularly applicable is the asymmetry of computational effort in the training and testing stages of the algorithm. Classifying new data points requires orders-of-magnitude less computation than training because the process of training an SVM requires solving a quadratic optimization problem.

SVM training requires O(n^3) operations, where n is the number of training examples. In contrast, testing a new vector with a trained SVM requires only O(n) operations. Faster training algorithms that exploit the specific structure of the SVM optimization problem exist, but the training remains the primary computational bottleneck.

After training the SVM, many of the weights, w_i, will be equal to zero. This means that these terms can be ignored in the classification formula. Input vectors that have a corresponding non-zero weight are called support vectors. Reducing the number of support vectors is key to successfully deploying an SVM classifier on board a spacecraft, where there are severe constraints on the amount of CPU resources available.

Previously deployed classifiers have used such reduced-set methods, but were still constrained to operate on only a subset of the available classification features. Removing such bottlenecks is critical to realizing the full potential of SVMs as an onboard autonomy tool.



```
for(class_index=0; class_index<NUM_CLASSES; class_index++) {

    PIXEL output =
    (-svm_biases[class_index]) +
    ref[8]   * svm_coefs[class_index][0] +
    ref[21]  * svm_coefs[class_index][1] +
    ref[31]  * svm_coefs[class_index][2] +
    ref[34]  * svm_coefs[class_index][3] +
    ref[41]  * svm_coefs[class_index][4] +
    ref[51]  * svm_coefs[class_index][5] +
    ref[85]  * svm_coefs[class_index][6] +
    ref[110] * svm_coefs[class_index][7] +
    ref[150] * svm_coefs[class_index][8] +
    ref[210] * svm_coefs[class_index][9] +
    ref[213] * svm_coefs[class_index][10];

    if (class_index==0 || output > best_value) {
        best_class = class_index;
        best_value = output;
    }

    co_stream_write(outband, &best_class, sizeof(best_class));
}
```

Figure 3 – Software/hardware partitioning for the SVM algorithm

Partitioning the Problem

When using FPGAs with embedded processors, efficient partitioning of algorithms between software and hardware is important to achieve high performance. For the FPGA-based development of the SVM, we implemented a previously software-only legacy algorithm in the FPGA hardware fabric to take advantage of the FPGA's high-speed parallel processing capabilities. The image file input and classification file output are managed within the embedded PowerPC processor using the CompactFlash card provided on the ML410 board.

In this implementation, the software side of the application is coded in C and compiled to the embedded PowerPC 405 processor using the Xilinx EDK tools. The embedded software application reads an input image file consisting of 857,856 pixels. The image file is read from the CompactFlash card installed in the ML410 board.

The software-side application streams the image data to the SVM, which is also written in C but has been compiled (using the Impulse C-to-FPGA compiler) to FPGA hardware. The SVM hardware process performs the required SVM operation on the image and streams the results back to the PowerPC 405 processor. The processor then writes the pixel classifications (e.g., snow, water, ice, land, cloud, or unclassified) to an output file on the CompactFlash card (Figure 3).

The PowerPC ran the software portion of the task, which sends data to and collects data from the SVM hardware module. We chose to use the PowerPC instead of a MicroBlaze™ processor because the PowerPC can operate at triple the clock frequency of the MicroBlaze processor. Also, the MicroBlaze processor would be instantiated in valuable FPGA fabric, whereas the PowerPC exists external to the fabric.

Because the 256-MB DIMM is the largest source of memory on the board, we used it as main memory for the program. The processor local bus (PLB) is a high-speed bus (compared to the on-chip peripheral bus [OPB]) that allows for fast data transfer to/from the memory and SVM core peripherals. The 16-GB CompactFlash card holds the input and output data files, which are too large to fit on the DIMM. The UART was used for debugging output. The OPB is a low-speed bus that serves as the default interface between the processor and the SystemACE™ interface controller and UART peripherals.

The Impulse C compiler performs these optimizations and generates hardware in the form of either VHDL or Verilog. This hardware can then be synthesized using FPGA tools such as Xilinx ISE™ software and Platform Studio.

In support of partitioned software/hardware applications such as this, the Impulse tools include a library of C-compatible functions that implement a number of process-to-process communication methods. These methods include streaming, shared memory, and message passing. For this application, the Impulse C streaming programming model was the obvious choice.

In Impulse C streaming applications, hardware and software processes communicate primarily through buffered data streams implemented directly in hardware. This buffering of data makes it possible to write parallel applications at a relatively high level of abstraction without the cycle-by-cycle synchronization that would otherwise be required.

Figure 4 illustrates the design flow for C-to-hardware compilation using the Xilinx FX60 FPGA as a target.

On the software side of the application (in this case on the PowerPC 405 processor used for hardware-level testing), Impulse C functions are used to open and close data streams, read or write data on the streams, and, if desired, send status messages or poll for results. In the case of the Virtex-4 FX, stream reads and writes can be specified as operations that take advantage of either the PLB or the auxiliary peripheral unit (APU) interface.

Generating Parallel FPGA Hardware

To create the hardware portion of our application, we used the Impulse C compiler to generate synthesizable HDL files ready to use with the Xilinx EDK tools. In addition to generating HDL files, the Impulse compiler also generates additional files required by the EDK tools, including the needed PLB and APU bus interfaces. The Impulse C compiler performs a variety of low-level optimizations, including C statement scheduling and loop pipelining, saving application developers a great deal of time that would otherwise be spent performing tedious, low-level hardware optimization.

The Impulse C compiler performs these optimizations and generates hardware in the form of either VHDL or Verilog. This hardware can then be synthesized using FPGA tools such as Xilinx ISE™ software and Platform Studio. On the processor side, the compiler generates run-time libraries ready for use on the embedded PowerPC processor.

Validating the Algorithm

The output of the combined software/hardware application is a file comprising a column of integers indicating the resulting class of each pixel in the image. To validate the results of the experiment and see the results, we used MATLAB to reformat the column of integer values to the original pixel-wise dimensions of the image. Each class was assigned an arbitrary color, and the number of pixels belonging to each class was tabulated. We could then easily calculate the percentage of pixels belonging to each class as well as visualize the resulting file of classified pixels as a colored image.

This validation was required to meet two goals of this project. First, it was necessary to validate both the pixel classification results from the legacy (software-only) version of the SVM and the generated hardware implementation of the SVM. This was necessary to verify that the integer and floating-point calculations performed in the FPGA (in hardware) returned the same results (within acceptable margins) as those observed in software currently flying.

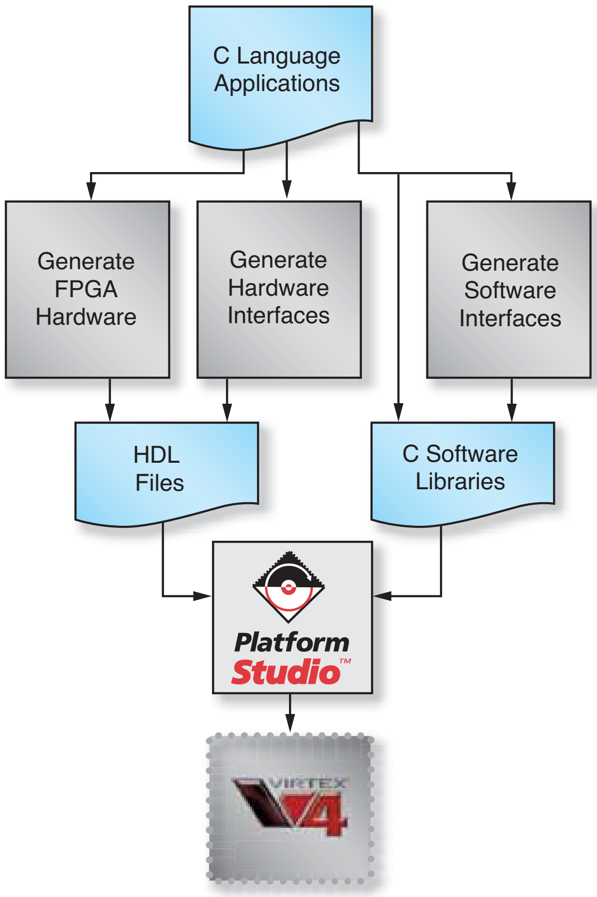
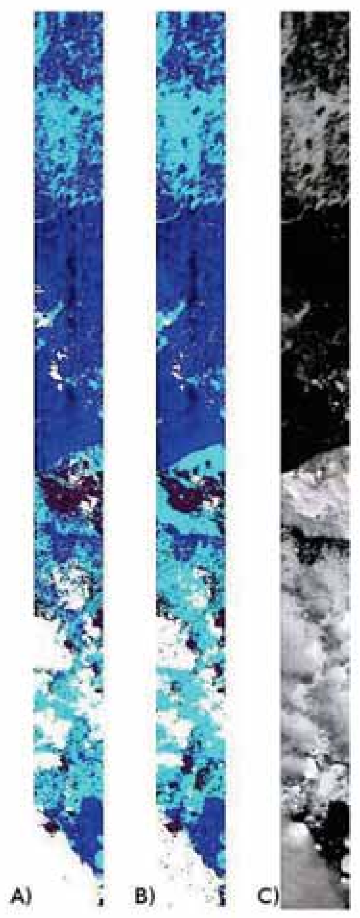


Figure 4 – Design flow from C-code to FPGA-embedded application



[The color key is blue = water, cyan = ice, dark purple = snow, lavender = unclassified]

Figure 5 – A comparison of the results from a) the legacy 11-band SVM implementation, b) the FPGA-accelerated implementation, and c) the original hyperspectral image.

We began the validation process by comparing the pixel classification percentage results to those reported by the SVM used in the currently flying EO-1 satellite. The classification percentages show good agreement, particularly for the snow and water classes.

Image visualizations were also important in this effort. Our resulting visualizations show excellent agreement with the results from the EO-1 (Figure 5).

In addition to the qualitative comparison of the images, we also conducted a pixel-by-pixel comparison of the legacy algorithm results and our classifications. The pixel-by-pixel classification comparison showed that 76.8% of the pixel classifications in our results matched those of the Autonomous Sciencecraft Experiment (ASE). We believed that the remaining discrepancies were caused by the differences in the training data sets of the SVMs, but nonetheless we decided to verify the hardware implementation on a pixel-by-pixel basis.

To completely validate the algorithm, we compared the outputs of a software implementation, using the same input data, to the version implemented in hardware by the Impulse C compiler. The two implementations produce identical classifications on a pixel-by-pixel basis. The combination of the good agreement of our results with the legacy ASE results, as well as the independent results from the software platform, led us to believe that our implementation was valid. All of these validations were performed using a combination of C language and MATLAB programming methods, with no need to use hardware design methods or hardware description languages.

Extending the Algorithm

Having successfully implemented the legacy SVM designed for Hyperion, we then considered two extensions to the C-language algorithm: using a larger number of bands with the same linear kernel SVM and creating a new SVM with a nonlinear kernel. For the expanded linear kernel SVM, we arbitrarily selected 30 of the available 242 bands in the image. For the non-linear kernel SVM, we used the same 11 bands as the legacy SVM, but with a modified kernel. Because training data was not available for the original legacy SVM, we could not generate new SVMs that would be comparable to it, so we used new training data to generate the two new SVMs. We also generated a new, 11-band linear-kernel SVM for comparison to the legacy SVM.

Using the C-to-hardware compiler, we were able to quickly experiment with these alternative implementations and compare results, both in terms of accuracy and in terms of performance. The hardware implementation of these SVMs produced results that agree very well with the software simulations of the algorithms and demonstrate significant increases in overall system performance.

We were also able to determine (using Xilinx synthesis tools) the FPGA fabric utilization percentages for each of these SVMs, as shown in Table 1.

Conclusion

FPGAs with embedded processors are demonstrating levels of performance and efficiency that were previously impossible using traditional processors. Hardware acceleration of SVM algorithms promises to dramatically improve onboard data processing in future science missions.

By using embedded FPGAs such as the Virtex-4 FX60 device, we can implement increasingly advanced SVMs with "room to grow" in onboard resources. Our results demonstrate that we can achieve even our most advanced SVM extension, a polynomial kernel, using just one-third of the DSP blocks and slices available in the Virtex-4 FX-60 FPGA.

Software-to-hardware design tools played an important role in the fast prototyping and development of these SVM algorithms. The Impulse C tools allowed us to more easily manage the complexity of the application and to experiment with alternative implementations. For testing and algorithm validation, the Xilinx ML410 board provided an excellent and cost-effective development target.

FPGA Resources	Total on V4FX60	Linear (11 bands)	Linear (30 bands)	(2,1) Polynomial
Slices	25,280	1,151 (4%)	2,253 (8%)	2,082 (8%)
Slice Flip-Flops	50,560	1,290 (2%)	1,337 (2%)	2,519 (4%)
Four-Input LUTs	50,560	1,838 (3%)	3,110 (6%)	3,287 (6%)
FIFO16/RAMB16s	232	2 (1%)	2 (1%)	5 (2%)
DSP48s	128	4 (3%)	4 (3%)	12 (9%)

Table 1 – Device utilization for an FPGA-based SVM algorithm