Fast Real-Time LIDAR Processing on FPGAs

K. Shih, A. Balachandran, K. Nagarajan, B. Holland, C. Slatton, A. George
NSF Center for High-Performance Reconfigurable Computing (CHREC)
Department of Electrical and Computer Engineering
University of Florida, Gainesville, FL

Abstract—*Light Detection and Ranging (LIDAR) plays an important role in remote sensing because of its ability to provide high-resolution measurements of 3D structure. For time-sensitive airborne missions, fast onboard processing of LIDAR data is desired and yet difficult to achieve with traditional embedded CPU solutions due to the computational requirements. FPGAs have the potential to speed up processing by employing multi-level parallelism, but their use in LIDAR processing has typically been limited to data capture due to the difficulties associated with efficiently migrating LIDAR processing algorithms to FPGAs. We demonstrate two equivalent FPGA designs for coordinate calculation of LIDAR data written using different languages (VHDL and MATLAB-based AccelDSP), comparing their performance and productivity. For the VHDL design, a ~14× speedup is obtained over an Opteron processor on a Cray XD1 system. In addition, a recently proposed performance prediction methodology is employed, and the accuracy of its pre-implementation predictions is analyzed.*

1. Introduction

Over the past few decades, airborne LIDAR technology has emerged as an important remote sensing modality for many scientific and military applications [1], [2]. Most LIDAR applications involve terrain mapping, but LIDAR data have also been fused with other sensor types, such as multi-spectral imagery [3]. The importance given to LIDAR data comes from its capabilities to provide high-resolution position information on targets of interest from a remote distance. These targets include terrain topography, vegetation structure, and building features. Obtaining high-resolution data is made possible through high density of laser returns. For example, Optech Gemini system records laser returns at frequencies as high as 167 kHz [1], [2], [4], collecting more than ten million laser returns every minute. Raw LIDAR data (laser ranges, scan angles, etc.) recorded by the sensor needs to be processed in order to present information in the form of a 3D point cloud, and such processing is computationally demanding at near realtime rates due to the high laser pulse rates. For example, compact modern ground-based LIDARs can record laser ranges for pulse rates up to a few kHz [5]. Discrete-return airborne LIDARs operate at laser pulse rates in excess of 150 kHz and record four or more returns per transmitted pulse [2]. Spaceborne LIDARs typically have lower pulse rates, but digitize the return pulse into 100 or more samples [6]. As a result, LIDAR data are often saved to onboard storage devices and processed off-line on PC workstations at a later time. However, in a time-constrained scenario, the acquired data have to be processed onboard for realtime analysis and feedback.

Onboard processing of LIDAR data may be feasible if LIDAR operators on the aircraft use commercially available laptops with large-volume hard disk drives, but many General-Purpose Processors (GPPs) on laptops still render tasks serially and thus likely fail to meet the realtime analysis requirements. In contrast to GPPs, High-Performance Embedded Computing (HPEC) systems featuring FPGAs can be used to speed up the procedure by exploiting multi-level parallelism inherent in algorithms used for LIDAR processing. Moreover, the reconfigurability of FPGAs opens the possibility to migrate diverse signal processing algorithms to hardware designs according to a particular application's requirements.

Although FPGAs have much to offer in terms of power, adaptivity, and performance improvements, developing efficient designs that function at high frequencies using conventional hardware description languages (HDLs) can be unwieldy for application scientists because the underlying programming techniques commonly require detailed hardware knowledge that can be beyond their comprehension or interest. In addition, short design times and more importantly shorter re-design times would increase the productivity of application scientists when migrating complex algorithms. Therefore, generating efficient hardware designs by translating high-level languages (HLLs) to HDLs using application mappers (HLL tools) is highly desirable.

It is also critical for application designs to meet the requirements of the project during the migration to hardware. Yet, different algorithmic approaches in combination with potential platform architectures can likely lead to dissimilar designs with distinct performance improvements (or degradations), and this makes it time-consuming and inefficient to determine the most favorable choice of algorithmic approach and platform architecture by developing full hardware designs. Therefore, it is important to estimate the likely outcome of a new design (speedup and resource usage) before expending significant effort on any specific algorithm, architecture, or platform. In other words, an efficient and accurate prediction methodology can likely increase performance and productivity while minimizing unnecessary development time and effort.

The remainder of this paper is structured as follows. In Section 2, previous works related to LIDAR processing on FPGAs are reviewed, followed by a discussion of the fundamentals of onboard LIDAR processing in Section 3. In

Section 4, the hardware design methodology is explained in detail. In Section 5, experimental results are reported, and performance and productivity results of the HLL- and HDL-based designs are compared. Concluding remarks are given in Section 6 by listing key insights gained and scope for future work.

2. Related work

In [7], the authors described the design of a new photon-counting LIDAR system that uses a number of Xilinx FPGAs. This type of LIDAR has a slower laser pulse rate than the Optech Gemini, but actually has a higher data rate because it pixelates the laser footprint. The FPGAs were primarily used for registering each photo-electron return event from a 10×10 multi-channel photo-multiplier tube (PMT), which constitutes the detector. Similar work was also found in [8] for a multi-kilohertz micro-laser altimeter developed by NASA Goddard Space Flight Center. At a laser pulse rate of 10 kHz, a 10×10 detector can generate 1,000,000 return events per second. The actual data rate can be even higher since each return pulse may be spread out in time and thus generate more than one return "event" in each channel. In [7] and [8], the FPGAs were simply used to record the raw ranges for each return laser pulse, and thus additional parallelism gains could be realized by designing architectures using FPGAs to efficiently process the LIDAR data into 3D locations. The determination of the 3D locations is referred to as the "coordinate calculation algorithm" in this work.

Numerous HLL tools have been developed to help application scientists less acquainted with HDLs generate hardware designs in HLLs like C and MATLAB. Ease-of-use and development time are commonly considered in evaluating the efficiency of HLL tools. Comparative studies of HLL tools on their productivity and capabilities have been investigated in [9]. However, the extent of parallelism extracted from the algorithm by an HLL tool and its ability to generate designs for multiple platforms are also critical. In contrast to side-by-side comparisons among HLLs, comparison to conventional HDLs can provide a different perspective.

In [10], an RC Amenability Test (RAT) was proposed to quickly predict performance in application design migration to FPGAs with reasonable accuracies. RAT predicts performance for a particular design on a specific platform by modeling the algorithm and the platform in terms of a set of parameters. Since it is based on simple analytical models, RAT also offers the possibility to efficiently explore different architectures for a specific algorithm. The RAT methodology is used in this work to analyze the expected performance of the coordinate calculation algorithm on multiple FPGA systems.

3. Application overview

Although LIDAR systems can be configured to operate in different environments such as air, space, or land, the airborne configurations are the most common for terrain mapping and are considered here. In general, an airborne LIDAR system contains the following major components:

- Pulsed laser
- Scanner and optics
- Receiver and receiver electronics
- Position and navigation systems

The laser emits laser pulses at a prescribed frequency into the scanner optics, which governs the direction of the laser pulses as they travel toward the targets. The receiver registers the laser photons reflected from the terrain and targets of interest. The distance between the aircraft and the target (range, ρ) can then be obtained by measuring the travel time of the laser pulse and multiplying it by the speed of light in the atmosphere. The scan angle (θ) from the nadir direction underneath the aircraft is also recorded. The IMUs (Inertial Measurement Units) typically used update the aircraft attitude (roll φ_r, pitch φ_p, and yaw φ_y angles) and integrate accelerations to determine position at many tens of Hz, while the onboard GPS (Global Positioning System) provides a better absolute solution for the aircraft position (X_{ac}, Y_{ac}, Z_{ac}) at a slower rate of 1–2 Hz. The GPS and IMU output are combined to produce an improved estimate of the aircraft's trajectory. An illustration of an airborne LIDAR system is shown in Fig. 1.

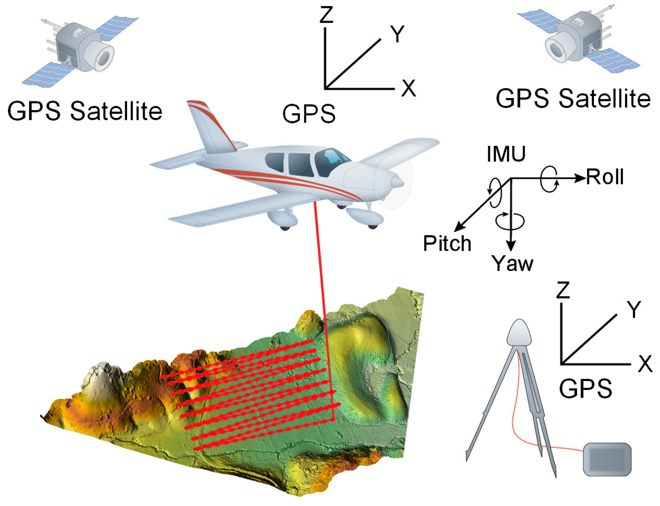

Fig. 1. Schematic illustration of an airborne LIDAR system

The fundamental computation in LIDAR processing is the calculation of coordinates of the laser returns using the eight LIDAR parameters, ρ, θ, φ_r, φ_p, φ_y, X_{ac}, Y_{ac}, and Z_{ac}. Each return's coordinate (X, Y, Z) is obtained by the following steps:

- Determining the unit vector that points from the sensor to the target for each laser pulse using scan angle (θ)
- Generating three rotation matrices that align the body-fixed vectors of the aircraft with Earth-fixed GPS coordinates using the respective angles (φ_r, φ_p, φ_y)
- Applying the three rotation matrices to the unit vector
- Scaling the rotated unit vector by the range value (ρ) to produce a range vector

- Translating the range vector to the Earth-fixed GPS coordinate frame using the GPS position (X_{ac}, Y_{ac}, Z_{ac}) and vector addition

Eq. 1 shows the resulting formula for coordinate calculation, in which C and S abbreviate cosine and sine operations, respectively.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \rho\left(C\varphi_y C\varphi_r S\theta - C\varphi_y S\varphi_r C\varphi_p C\theta - S\varphi_y S\varphi_p C\theta\right) + X_{ac} \\ \rho\left(S\varphi_y C\varphi_r S\theta - S\varphi_y S\varphi_r C\varphi_p C\theta + C\varphi_y S\varphi_p C\theta\right) + Y_{ac} \\ \rho\left(-S\varphi_r S\theta - C\varphi_r C\varphi_p C\theta\right) + Z_{ac} \end{bmatrix} \quad (1)$$

The LIDAR parameters in the calculations are "multi-rate," meaning that different parameters are updated at different frequencies. In particular, IMU and GPS components update at relatively slower frequencies compared to the laser pulse frequency. This behavior mandates an interpolation operation on the aircraft attitude and position values between their consecutive updates before being used for coordinate calculation.

Although laser returns are independent of one another and thus can be processed in parallel, the LIDAR parameters updated at higher frequencies need to be temporarily stored while awaiting the next available parameter that is updated at the slowest frequency. Fig. 2 illustrates the concept of how such multi-rate parameters can be processed in parallel using FPGAs. A pair of buffers is placed between the incoming stream of LIDAR parameters and the LIDAR processor on the FPGA. In an alternating fashion, one buffer receives data from the incoming stream while the other feeds previously stored data to the FPGA for processing. Switching between the two buffers occurs when the receiving one is full and contains at least one new value of the parameters that are updated at the slowest frequency. In addition to coordinate calculation, the processing procedure can potentially include additional stages before the occurrence of buffer switching, assuming sufficient FPGA resources are available. For example, interpolating the 3D points from the terrain into a continuous elevation image, known as a Digital Elevation Model (DEM), is commonly used for visual interpretation and could be migrated to an FPGA implementation in the future.

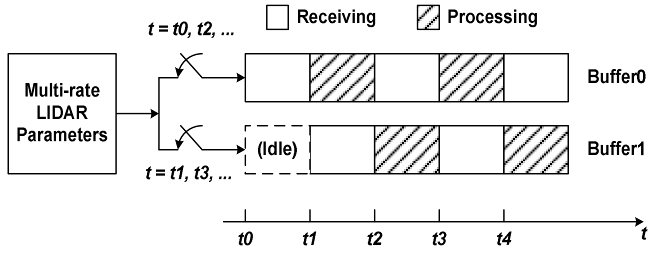

Fig. 2. Conceptual illustration of batch-processing of LIDAR data: Whether streaming, multi-rate LIDAR parameters are stored in Buffer0 or Buffer1 depends on the two switches governed by time t. When $t = t0, t2, \ldots$, Buffer0 receives data and Buffer1 processes data, and vice versa when $t = t1, t3, \ldots$.

4. Design methodology

An emulation of onboard LIDAR processing was realized by migrating a MATLAB-based LIDAR simulator to the FPGA. The LIDAR parameter settings are listed as follows:

- Laser pulse rate: 33 kHz
- Scan angle reading rate: 33 kHz
- IMU update rate: 5 Hz
- GPS update rate: 1 Hz

The buffer size is assumed to be able to contain one second of LIDAR parameters, which include 33,000 laser returns, 33,000 scan angle values, five sets of aircraft attitude values, and one GPS position. Furthermore, pre-design analyses were performed with respect to numeric precision and predicted performance. First, several fixed-point configurations were tested to determine whether a fixed-point implementation is suitable for LIDAR processing by studying resulting errors in the coordinate values. Second, RAT was used for estimating preliminary performance and also aided in design progression in this work. Precision analysis, RAT analysis, and the architecture designs are described in detail in the following sub-sections.

4.1. Precision analysis

Using MATLAB's fixed-point toolbox, a precision analysis was performed on a set of LIDAR parameters for 33,000 laser returns generated by the LIDAR simulator. Among LIDAR parameters, angular values (θ, φ_r, φ_p, φ_y) require more fractional precision than position values (ρ, X_{ac}, Y_{ac}, Z_{ac}, X, Y, Z). Table I shows several fixed-point configurations for LIDAR parameters and the corresponding increase in error versus double-precision floating point. The first pair of parentheses represents the fixed-point configuration for angular values, and the second pair represents position values. The two numbers inside the parentheses denote the number of total bits and the number of fractional bits, respectively. Errors caused by the conversion from floating point to fixed point are measured in terms of root-mean-squared error (RMSE) and maximum error (Max E), all of which are less than an acceptable margin of one meter, thus justifying the use of fixed-point precision in coordination calculation for LIDAR processing.

TABLE I
ERRORS MEASURED IN PRECISION ANALYSIS

Precision	RMSE (m)	Max E. (m)
(16, 14) & (16, 5)	0.254	0.962
(31, 28) & (16, 5)	0.183	0.718

4.2. RAT analysis

RAT analysis is performed in the form of a worksheet, as shown in Table II. An "element," which corresponds to a laser return in this application, is examined to estimate two performance-indicative quantities: time taken to transfer data in and out of the FPGA, termed as communication time (t_{comm}), and time taken to perform all the algorithmic computations on the transferred data, termed as computation time (t_{comp}). It is common for application data to be significantly larger than the available FPGA system memory, and thus the data set may need to be broken into smaller blocks and processed on the FPGA.

N_{iter} denotes the number of iterations required to process all application data. The total time spent on the FPGA (t_{RC}) is then compared to a software baseline (t_{soft}) to obtain a predicted speedup. The entries in Table II that are related to estimating t_{comm} are (i) number of elements transferred per iteration ($N_{elements}$), (ii) element size ($N_{bytes/element}$), and (iii) ideal interconnect throughput with efficiency factors considered ($Throughput_{ideal}$, α_{read}, α_{write}). The entries related to t_{comp} are (i) $N_{elements}$, (ii) number of operations to be performed on each element ($N_{ops/element}$), (iii) number of operations that can be performed within one clock cycle ($Throughput_{proc}$), and (iv) assumed FPGA working frequency (f_{clock}). The set of formulae in Eq. 2 highlights the relationship between the estimates and the parameters considered.

$$\begin{aligned}
t_{read/write} &= \frac{N_{elements} \times N_{bytes/element}}{\alpha_{read/write} \times Throughput_{ideal}} \\
t_{comm} &= t_{read} + t_{write} \\
t_{comp} &= \frac{N_{elements} \times N_{ops/element}}{f_{clock} \times Throughput_{proc}} \\
t_{RC} &= N_{iter}(t_{comm} + t_{comp}) \\
Speedup &= \frac{t_{soft}}{t_{RC}}
\end{aligned} \tag{2}$$

TABLE II
DESIGN PROGRESSION THROUGH RAT ANALYSIS

Data Set Parameters		Design 1 (Nallatech)	Design 1 (Cray)	Design 2 (Cray)
$N_{elements}$, input	(elem.)	66000	66000	33018
$N_{elements}$, output	(elem.)	33000	33000	33000
$N_{bytes/element}$	(B/elem.)	8	8	8

Communication Parameters		Design 1 (Nallatech)	Design 1 (Cray)	Design 2 (Cray)
$Throughput_{ideal}$	(MB/s)	1000	1638.4	1638.4
α_{read}	$0<\alpha<1$	0.25	0.5	0.5
α_{write}	$0<\alpha<1$	0.25	0.5	0.5

Computation Parameters		Design 1 (Nallatech)	Design 1 (Cray)	Design 2 (Cray)
$N_{elements,}$ proc	(elem.)	33000	33000	33000
$N_{ops/element}$	(ops/elem.)	9	9	10
$Throughput_{proc}$	(ops/cycle)	9	9	10
f_{clock}	(MHz)	125	125	125

Software Parameters		Design 1 (Nallatech)	Design 1 (Cray)	Design 2 (Cray)
t_{soft}	(sec)	1.09E-02	1.09E-02	1.09E-02
N_{iter}	(iter.)	1	1	1

Calculated Metrics		Design 1 (Nallatech)	Design 1 (Cray)	Design 2 (Cray)
t_{comm}	(sec)	3.16E-03	9.67E-04	6.45E-04
t_{comp}	(sec)	2.64E-04	2.64E-04	2.64E-04
t_{RC}	(sec)	3.43E-03	1.23E-04	9.09E-04
Predicted Speedup		**3.2**	**8.9**	**12.0**

Table II also summarizes the design progression aided by predictions made through RAT. Two platforms were considered in RAT analysis: first, a 16-node Linux cluster with each node configured with a 3.2 GHz Intel Xeon processor and a Nallatech H101 PCI-X board featuring a Xilinx Virtex4 LX100 FPGA (only one node was used for LIDAR designs) and second, a Cray XD1 machine with six nodes in one chassis, each featuring two 2.4 GHz AMD Opteron processors and one Xilinx Virtex2 Pro 50 FPGA (only one node was used for LIDAR designs). For Host-to-FPGA communication, the Nallatech board uses a PCI-X bus, while the Cray XD1 is equipped with a RapidArray interconnect. The baseline t_{soft} was computed from a baseline C application executed on a 2.4 GHz AMD Opteron processor with single-precision floating point. The FPGA working frequency f_{clock} is assumed to be 125 MHz.

Two designs, Design 1 and Design 2, were considered and analyzed during this design progression. Design 1 assumes the LIDAR parameters updated at slower rates are interpolated up to 33,000 times on the Host, and the computation on the FPGA involves only coordinate calculation. A 16-bit fixed-point configuration was chosen for LIDAR parameters and the resulting coordinates. Although a total of 48 bits are sufficient to represent one set of coordinates (X, Y, Z) for a laser return, these values are byte-packed to 32 or 64 bits ($N_{bytes/element} = 8$) to match the interconnect bandwidth (Nallatech: 32-bit; Cray XD1: 64-bit). Byte-packing was applied to LIDAR parameters as well. Byte-packed LIDAR parameters require 128 bits per element, while the resulting coordinates require 64 bits per element. Therefore, $N_{elements}$ for input is twice $N_{elements}$ for output. Since each laser return can be processed independently, a 9-cycle-long pipeline was constructed based on Eq. 1 ($N_{ops/element} = 9$). Furthermore, the number of parallel pipelines is set to one in this application due to the assumption that the FPGA may not have sufficient bandwidth to support multiple pipelines and hence determines the number of operations per cycle ($N_{ops/cycle} = 9$). FPGA memory is assumed to have capacity of one full buffer of LIDAR parameters ($N_{iter} = 1$), and $N_{elements}$ for processing is the same as $N_{elements}$ for output as an element is referred to a laser return in this application.

As shown by the calculated metrics in Table II, most of t_{RC} is dominated by t_{comm} (~92% of t_{RC}) when considering the Nallatech board the target platform. RAT indicates that the coordinate calculation stage in LIDAR processing is a communication-bound process under Design 1, and therefore, the Cray XD1 platform is better suited for this application design. The higher effective throughput of the Cray XD1 (819.2 MB/s, as opposed to 250 MB/s of the Nallatech board) is estimated to reduce t_{comm} significantly (~79% of t_{RC}). Moreover, an emerging trend in FPGA technologies is to architecturally integrate CPU, memory, and FPGA resources at the system level as opposed to traditional peripheral-bus interfaces [11]. The Cray XD1 is representative of such systems and thereby an appropriate choice for the purpose of prototyping embedded applications like onboard LIDAR processing.

Migration of LIDAR processing algorithms was re-investigated in order to further reduce t_{comm}. Design 2 attempts to balance communication and computation by migrating interpolation for parameters updated at slower frequencies to the FPGA. A linear interpolation method was realized by accumulating pre-calculated increments to base values comprised of un-interpolated parameters. The pseudo-code shown in Fig. 3 illustrates the steps for interpolating roll angles (φ_r) as an example. Similar steps are applied to other un-interpolated parameters. In Design 2, there are still 33,000 range and scan angle values (ρ, θ) being transferred from the Host to the FPGA, but only five sets of (φ_r, φ_p, φ_y) with their increments ($\Delta\varphi_r$, $\Delta\varphi_p$, $\Delta\varphi_y$) and one set of (X_{ac}, Y_{ac}, Z_{ac}) with (ΔX_{ac}, ΔY_{ac}, ΔZ_{ac}) are sent to the FPGA. This reduction in number of parameters to be transferred enables the angular values (θ, φ_r, φ_p, φ_y) to increase the total bits used for fixed-point configurations since this increase only fills up the extra bits that would have been sent due to the interconnect bandwidth and thus the precision errors are further reduced by 28% in RMSE and 25% in Max E, respectively. In particular, a (31, 28) configuration was used in Design 2 for (θ, φ_r, φ_p, φ_y) and ($\Delta\varphi_r$, $\Delta\varphi_p$, $\Delta\varphi_y$). After byte-packing for Cray XD1's 64-bit interconnect, (ρ, θ) requires 64 bits per element, and (φ_r, φ_p, φ_y, X_{ac}, Y_{ac}, Z_{ac}) along with their increments require 18 64-bit transfers for all the elements. Therefore, compared to Design 1, $N_{elements}$ for input is reduced to 33,018, t_{comm} is decreased to ~71% of t_{RC}, and both $N_{ops/elements}$ and $Throughput_{proc}$ are increased by 1 since the pipeline is extended one extra cycle to accommodate the accumulation of the increments. Given these changes, executing Design 2 on Cray XD1 platform was predicted to achieve a speedup factor of 12 over the baseline executed on an Opteron processor.

```
/* The Host side */
for i = 1 to 5
    $\Delta\varphi_r(i) = [\varphi_r(i+1) - \varphi_r(i)]/[33000/5]$; /* pre-calculate $\Delta\varphi_r(i)$ */
end
```

```
/* The FPGA side */
for i = 1 to 5
    $\tilde{\varphi}_r = \varphi_r(i)$; /* load new base value */
    for j = 1 to (33000/5)
        $\tilde{\varphi}_r = \tilde{\varphi}_r + \Delta\varphi_r(i)$; /* accumulate increment to base value */
        $(X, Y, Z) \leftarrow f(\ldots, \tilde{\varphi}_r, \ldots)$; /* coordinate calculation using $\tilde{\varphi}_r$ */
    end
end
```

Fig. 3. Pseudo-code illustrating the steps for interpolating the roll angles on the Host side and the FPGA side

4.3. Architecture design

Both Design 1 and Design 2 contain a LIDAR processing core that computes coordinates and a state machine that governs the data flow, as shown in Fig. 4. The processing core is fully pipelined by extracting the parallelism inherent in coordinate calculation, including independent computation of the coordinates (X, Y, Z) and simultaneous sinusoidal evaluations of the angular values (θ, φ_r, φ_p, φ_y). The Cray XD1 system provides several ways of bidirectional data transfer between the Host and the FPGA [12]. One is Host-initiated transfer through the use of API functions. Another is FPGA-initiated transfer involving the use of both API functions and a DMA transfer module. The former was found inefficient in transferring data from the FPGA to the Host [13], and hence the latter was considered to exploit the high throughput efficiency of the RapidArray interconnect. Thus, data can be sent from the QDR-II SRAM on the FPGA to the Host memory using a DMA transfer, bypassing the Host processor.

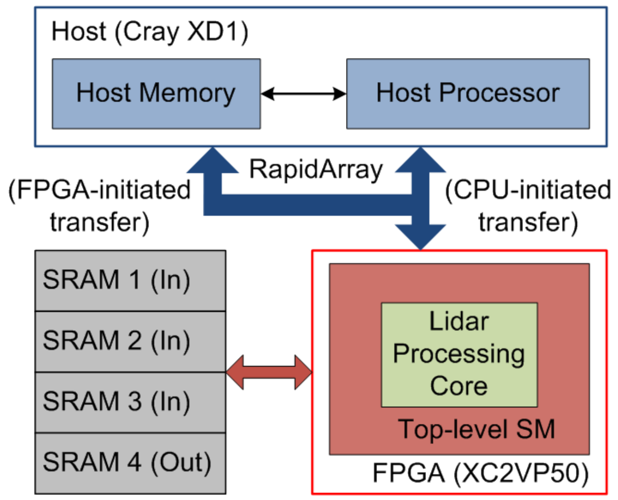

Fig. 4. Data flow of onboard LIDAR processing on Cray XD1 with DMA

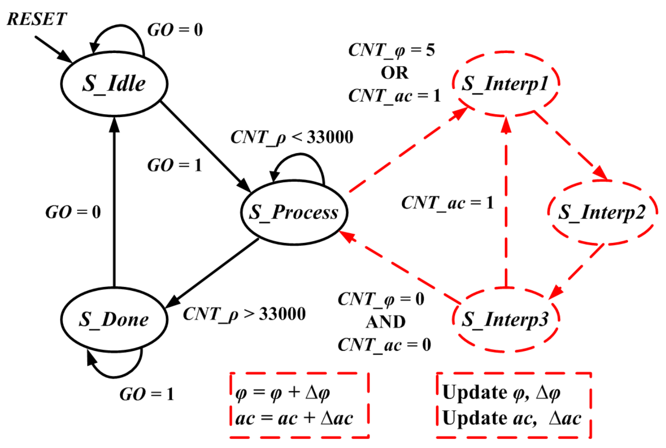

Fig. 5. State machine diagram (Design 1: states with solid lines; Design 2: states with solid lines and states with dashed lines)

The design progression from Design 1 to Design 2 incurs no architectural changes to the processing core but a modification on the state machine. The state machine in Design 1 is composed of three states: *S_Idle*, *S_Process*, and *S_Done*. At *S_Idle* the FPGA is reset and awaits a "start" signal ("*GO = 1*" in Fig. 5) from the Host. At *S_Process* the coordinates of laser returns are computed until all the transferred data are processed ("$CNT\_\rho > 33000$" in Fig. 5). At *S_Done* the FPGA awaits an "end" signal ("*GO = 0*" in Fig. 5) from the Host. The state machine in Design 2 consists of the same states as in Design 1 and three additional states: *S_Interp1*, *S_Interp2*, and *S_Interp3*, taking the parameter interpolation into account. As shown in Fig. 5, solid circles represent the common states while dashed circles represent the additional states. The three components of the aircraft attitude are stored sequentially in consecutive entries of the same bank of SRAM and thus need to be interpolated individually since the FPGA can only access one entry of the same bank of SRAM per clock cycle. The same interpolation procedure and state transitions apply to the components of the GPS position. Whenever a base value and its increment need to be updated ("*CNT_φ = 5 OR CNT_ac = 1*" in Fig. 5), the state machine transitions from *S_Process* to *S_Interp1*, goes through *S_Interp2* and *S_Interp3* sequentially, and then transitions back to *S_Process*.

5. Results

After a suitable architecture was determined, both Design 1 and Design 2 were developed and mapped to an FPGA on the Cray XD1 for analysis and verification of performance factors. Moreover, each design had two versions of its processing core developed, one using a MATLAB-based HLL tool (AccelDSP) and the other using conventional VHDL. These two versions of cores were encapsulated by the state machine and platform wrapper (developed in VHDL) in each design. All designs were synthesized in Xilinx ISE 9.1 to generate the final bitstream. Speedup and FPGA resource utilization are reported and discussed, followed by a comparison of performance between the two versions for each design.

5.1. Speedup and resource utilization

One primary goal of this work is to achieve performance improvements in terms of shorter processing time. With the FPGA running at a clock frequency of 125 MHz, which was the highest frequency obtained after repeated Synthesis and PAR procedure, speedups were obtained using the formula shown in Eq. 2 with the same software baseline and t_{RC} experimentally measured on the Cray XD1 system. Selected resource utilization values and obtained speedups are shown in Table III. Speedups of approximately 10× and 13× were achieved by Design 1 and Design 2, respectively, both deviating from the RAT predictions by less than 15%. The measured t_{comm} (Design 1: 0.659 ms; Design 2: 0.565 ms) was less than the RAT-projected t_{comm} because a conservative interconnect efficiency factor was considered in RAT. Thus, there was a slight increase in actual speedup values over their respective RAT projections. Design 2 involved more computation and a larger state machine than Design 1 and hence consumed more slices. As reported in Table I, the values obtained by executing Design 1 with fixed-point configurations of (16, 14) & (16, 5) and Design 2 with (31, 28) & (16, 5), respectively, on the Cray XD1 result in an RMSE less than 0.3 meters. In other words, it represents less than 3% error for a DEM over an area with 10 meters of topographic relief, or less than 0.3% error for a 100-meter relief. While some applications can require greater precision, that is an acceptable level for many applications.

TABLE III
SPEEDUP & DEVICE UTILIZATION OF DESIGNS 1 AND 2 ON CRAY XD1

Description		Design 1		Design 2	
		HLL	HDL	HLL	HDL
Slices	(%)	38	31	45	42
MULT18x18s	(%)	5	5	5	5
Actual Speedup		**9.9**	**10.2**	**13.1**	**13.8**

5.2. HLL vs. HDL comparison

Shorter development time is one of the most attractive advantages in using HLL tools for application scientists who are less experienced in HDL programming. In particular, HLL tools that aid in developing hardware designs by translating MATLAB codes to HDLs draw attention to researchers in signal processing fields, where MATLAB is one of the more popularly used programming languages. In this work, two researchers participated in developing application designs. One of them was a signal processing researcher and used AccelDSP to assist with design development, whereas the other was more familiar with hardware designs and developed the application in VHDL. The development time using VHDL, including the researcher's algorithm-acquaintance period, was roughly three times longer than using AccelDSP.

The primary parallelism inherent in coordinate calculation was automatically extracted and pipelined by AccelDSP in the HLL version and manually exploited in the HDL version. The resulting clock frequency, pipeline length, and selected resource utilization of the processing cores between the two versions are shown in Table IV. As expected, the HDL version has higher achievable core frequency because the automated procedure of translating HLL to HDL creates overhead that increases critical-path delays and thus results in a sub-optimal design. However, the actual board frequencies were limited by the state machine and the platform wrapper so that similar speedup values and negligible degradation were observed in comparison with the HDL design. The same number of multipliers used in both versions demonstrates that AccelDSP was able to match the efficiency of a hand-coded design with respect to these relatively scarce and expensive resources. Two essential techniques were useful for extracting parallelism inherent in the algorithm of coordinate calculation: loop-unrolling of matrix multiplications and pipelining for underlying data flow, both of which are frequently used for optimizing hardware designs. As both designs resulted in similar pipeline length, AccelDSP exhibited potential for exploiting parallelism through pipelining.

TABLE IV
HLL VERSUS HDL COMPARISONS IN CORE DESIGN

Core Design	Design 1		Design 2	
	HLL	HDL	HLL	HDL
Core Freq. (MHz)	140	180	150	200
Pipeline Len. (cycle)	31	34	31	34
MULT18x18s (%)	5	5	5	5
Slices (%)	18	17	20	17

HLLs are generally considered less efficient in manipulating bit-level operations than HDLs, but AccelDSP compensates for this efficiency loss by providing convenient constructs for performing such manipulations on components in designs while developing them using MATLAB [14]. Vendor-supported IP is also useful in accelerating HLL hardware designs. In this work, several essential sinusoidal evaluations were developed using such IPs [15]. AccelDSP also supports features for common parallelism-extracting techniques with a user-friendly interface. For instance, pipeline registers can be explicitly inserted before and/or after any operations to shorten long propagation delays.

Another beneficial feature of AccelDSP is its automated and flexible floating-to-fixed-point conversion, which can be inefficient and time-consuming to approach manually. This feature was frequently employed for result verification in this work since the default data format in MATLAB is double-precision floating point.

A common drawback of HLL tools is their requirement of a specific programming style to allow efficient extraction of parallelism and generation of optimized designs. AccelDSP requires the design function call (the core function being translated to HDL) to be positioned inside a loop in a script file that governs the data sent in and out of the core [16]. Although this restriction significantly increases AccelDSP's capability of optimizing pipelined designs, such as the core design for coordinate calculation in this work, it also limits the flexibility of modifying the top-level data flow, such as occurred with the addition of parameter interpolation in Design 2.

Although HLL tools like AccelDSP are helpful in developing application cores, platform wrappers still play an unavoidable role in the completion of a hardware design. They are usually available in conventional HDL and may not appeal to application scientists. It is a rising trend in the evolution of HLL tools that platform wrappers continue to provide more abstract interfaces so that developing a hardware design would entail much less knowledge of hardware details and HDL coding. However, the underlying non-conventional HLLs used in such tools may incur extra language-acquaintance effort.

6. Conclusions

Two different designs of hardware architecture were developed, mapped, and analyzed on the Cray XD1 system with Xilinx Virtex2 Pro 50 FPGAs. A final speedup factor close to 14 was achieved. RAT was used to examine application designs and predicted performance with variation less than 15%. RAT also indicated that coordinate calculation in LIDAR processing is a communication-bound process and thus motivated design progression in terms of choosing a suitable platform and modifying the algorithm for greater speedups. In addition, both HLL tools and HDL were used for the development of the processing cores, and the performance and productivity of the two versions were analyzed and compared. Minimal performance degradation from the HLL version was experienced because the final FPGA clock frequency was limited by the top-level state machine and the platform wrapper. HLL tools, which have a variety of convenient features, enable efficient migration of applications to hardware by reducing time and effort for development. Restrictions in programming style of HLLs may aid automated design optimization but also limit design flexibility.

As an extension of this work, subsequent signal processing components in LIDAR processing after the coordinate calculation stage could increase the amount of computation performed on the FPGA so long as FPGA resources are available. If a single FPGA does not contain sufficient resources, the use of multiple FPGAs should be investigated, along with the current design's scalability and the associated performance prediction techniques needed for multi-FPGA analysis. Moreover, the additional effort required for porting an AccelDSP-generated core to other platforms to perform the same computation could be an evaluating factor of its productivity as an HLL tool.

7. Acknowledgements

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422. The authors gratefully acknowledge vendor equipment and/or tools provided by Xilinx, Cray, Nallatech, Aldec, and Synplicity that helped make this work possible.

8. References

[1] W. Carter, R. Shrestha, and C. Slatton, "Geodetic laser scanning," *Physics Today*, Dec., 2007, pp. 41–47.

[2] C. Slatton, W. Carter, R. Shrestha, and W. Dietrich, "Airborne laser swath mapping: achieving the resolution and accuracy required for geosurficial research," *Geophysical Research Letters*, vol. 34, L23S10, doi:10.1029/2007GL031939, 2007.

[3] M. Mutlu, S. Popescu, C. Stripling, and T. Spencer, "Mapping surface fuel models using LIDAR and multispectral data fusion for fire behavior," *Remote Sensing of Environment*, vol. 112, 2008, pp. 274–285.

[4] Optech, "ALTM Gemini 167 brochure," 2006, http://www.optech.ca/pdf/Gemini167.pdf.

[5] C. Slatton, M. Coleman, W. Carter, R. Shrestha, and M. Sartori, "Control methods for merging ALSM and Ground-based laser point clouds acquired under forest canopies," *Proc. SPIE, 4th International Asia-Pacific Environmental Remote Sensing Symposium*, Honolulu, Hawaii, Nov., 2004, vol. 5661, pp. 96–103.

[6] J. Blair and M. Hofton, "Modeling laser altimeter return waveforms over complex vegetation using high-resolution elevation data," *Geophysical Research Letters*, vol. 26, 1999, pp. 2509–2512.

[7] W. Carter, R. Shrestha, and C. Slatton, "Photon-counting airborne laser swath mapping (PC-ALSM)," *Proc. SPIE, 4th International Asia-Pacific Environmental Remote Sensing Symposium*, Honolulu, Hawaii, Nov., 2004, vol. 5661, pp. 78–85.

[8] W. Powell, E. Hicks, M. Pinchinat, P. Dabney, J. McGarry, and P. Murray, "Reconfigurable computing as an enabling technology for single-photon-counting laser altimetry," *Proc. IEEE Aerospace Conference*, Big Sky, MT, Mar. 6–13, 2004, pp. 2327–2339.

[9] E. El-Araby, M. Taher, M. Abouellail, T. El-Ghazawi, and G. Newby, "Comparative analysis of high level programming for reconfigurable computers: methodology and empirical study," *Proc. IEEE III Southern Conference on Programmable Logic (SPL2007)*, Mar del Plata, Argentina, Feb. 26–28, 2007.

[10] B. Holland, K. Nagarajan, C. Conger, A. Jacobs, and A. George, "RAT: A methodology for predicting performance in application design migration to FPGAs," *Proc. High-Performance Reconfigurable Computing Technology and Applications Workshop (HPRCTA 2007)*, SC'07, Reno, NV, Nov. 11, 2007.

[11] P. Leong, "Recent trends in FPGA architectures and applications," *Proc. 4th IEEE Symposium on Electronic Design, Test and Applications (DELTA 2008)*, Hong Kong, SAR, China, Jan. 23–25, 2008, pp. 137–141.

[12] Cray, "Cray XD1 FPGA development," May, 2006.

[13] D. Chavarría-Miranda and A. Márquez, "Assessing the potential of hybrid HPC systems for scientific applications: a case study," *Proc. ACM International Conference on Computing Frontiers (CF'07)*, Ischia, Italy, May 7–9, 2007.

[14] Xilinx, "AccelDSP synthesis tool user guide," Sep., 2006.

[15] Xilinx, "AccelWare™ DSP IP toolkits user guide," Sep., 2006.

[16] Xilinx, "MATLAB for synthesis style guide," Sep., 2006.