FPGA-Based HPC Application Design for Non-Experts

David Uliana, Krzysztof Kepa, and Peter Athanas
Virginia Polytechnic Institute and State University,
Blacksburg, VA 24061, USA
{duliana,kepa,athanas}@vt.edu

***Abstract*—This work presents bFlow, an FPGA development framework for the rapid prototyping and implementation of hardware accelerators for hybrid computing platforms. This framework makes use of an abstracted, graphical front-end usable by those without computer engineering backgrounds, as well as an accelerated back-end that reduces compilation times, increasing design turns-per-day. bFlow's performance, usability, and application to the acceleration of big-data life-science problems verified by participants of the NSF-funded Summer Institute organized by the Virginia Bioinformatics Institute (VBI). In roughly one week, a group of four non-engineering participants made modifications to a reference Smith-Waterman implementation, adding functionality and scaling throughput by a factor of 4 to 600 million base pairs per second.**

I. Introduction

In the era of "data everywhere," information has gone from scarce to incredibly abundant. For instance, decoding the genome of a single human individual involves the analysis of three billion DNA base pairs. The prevalence of valuable and complex datasets of this sort is increasing at a rising rate. When the Sloan Digital Sky Survey (SDSS) began collecting data in 2000, it amassed more information in its first few weeks than all the data collected in the history of astronomy [1]. At a rate of about 200 GB per night, SDSS has amassed more than 140 terabytes of information. Its successor, the Large Synoptic Survey Telescope, anticipated to come online in 2016, is expected to acquire that amount of data every five days [2]. In 2010, main detectors at the Large Hadron Collider (LHC) produced 13 petabytes (13×10^{15} bytes) of data [3]. In these examples, "big data" refers to high complexity in addition to big volume. This trend is growing and creates the need for exceptional computing performance in order to efficiently process large quantities of data within tolerable times.

On the forefront of big-data computation are domain experts, e.g. bioinformaticians who explore novel genome hypotheses by analyzing massive amounts of data [4]. Such domain experts may be skilled programmers; however, their productivity–hypothesis discovery and verification rate–is limited by the performance limitations of available computing resources. The architecture of data-center class computing platforms, even high-performance servers or clusters, is designed for speed over a general mix of problems, and tailoring such platforms to domain-specific data structures and algorithms, such as DNA/protein sequence alignment [5], in an efficient, high performance manner, is not straightforward.

Heterogeneous computing machines like the Convey Hybrid-Core (HC) servers have potential to address this gap. These platforms enable the creation of application-specific accelerators that are tightly coupled with general-purpose processing resources. This tight coupling is achieved through the use of custom instructions and cache-coherent, shared memory space [6]. Unfortunately, development flows for such accelerators require extensive computer engineering and digital design expertise, including practical knowledge of a hardware description language such as Verilog and VHDL. These prerequisites are prohibitive for most domain-experts, including most experts in the life sciences community.

In this work, a development flow is proposed for hybrid computing systems like the Convey Hybrid-Core platforms. This approach aids non-engineers from the life sciences community in their creation of dedicated hardware accelerators, with the focus of avoiding the computational bottlenecks typical in their field. Consequently, the proposed flow should help such domain experts address '-omics' problems on an unprecedented computational scale [4].

Graphical design environments often provide a more intuitive mapping to parallel hardware than text- based languages; hence, a graphical desktop tool was selected as the design front-end of this flow. Specifically, DataIO's Azido (Figure 1) was chosen to fill this role. The accelerator designs are described in Azido in algorithmic form built on simple canonical primitives. Furthermore, the environment is structured in a manner that facilitates design targeting a variety of computational resources (e.g. FPGAs, CPUs, GPUs) using a single environment.

To target the Convey HC architecture, an Azido System Description was prepared. This architecture description acts as a plugin to Azido, encapsulating specifications of the platform organization and details about each of the core interfaces–namely the custom instruction dispatch, memory, and platform management interfaces. Furthermore, some interfaces are abstracted to simpler, higher level structures to ease development. For example, high throughput access to the Convey co-processor memory are abstracted to data stream sink and source objects defined within the system description.

All of these abstractions were complemented with software helper routines (C/C++) that abstract the co-processor API into a framework more intuitive to the big-data user.

The proposed approach relies on the following assumptions:

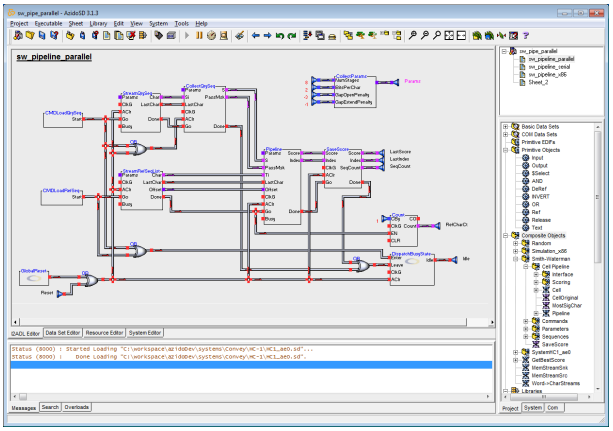

Fig. 1. The Azido graphical algorithm description environment.

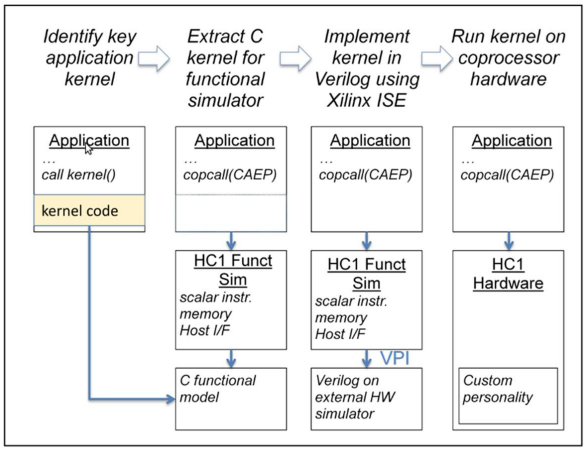

Figure 6 – PDK Design Flow

Fig. 2. Overview of Convey's Personality Development Kit, a flow for the development of custom personalities [11].

- Convey co-processor FPGA resources are the largest devices in the Xilinx Virtex-5 and Virtex-6 families, exceeding the requirements of many accelerators. Thus, some space and performance could be readily sacrificed for the sake of increasing the usability of the design tools.
- Azido's graphical, Implementation-Independent Algorithm Description Language (I2ADL) exposes the details of an algorithm while hiding the underlying implementation complexities. This core syntax, building on simplified system description abstractions, is intuitive and simple to the extent that a non-engineer domain expert with general-purpose programming expertise can use it to design hardware logic.

The rest of the paper is organized as follows. Section II provides some background and discusses the need for usable design environments targeting big-data hybrid-core computing platforms, as well as existing platforms and tools, Section III contains the approach taken in this work to address this need, and Section IV presents both qualitative and quantitative results of this effort. Conclusions and future work are given in Section V.

II. Background

Design productivity for FPGA-based computing has suffered from the contemporary ASIC "design productivity gap" and has unique needs and opportunities not adequately addressed by existing FPGA design tools. In [7] Nelson et al. proposed a productivity model that exposes three key contributors to high design productivity: multi-level design reuse, high-level design abstractions, and a more interactive verification environment which increases the number of development turns per day. All of these are necessary to improving design productivity, e.g. the use of high level (above HDL) languages would significantly reduce functional simulation time, thereby increasing turns per day. Also, describing designs in a hierarchical, high level language can promote reuse by making designs more portable.

High-level synthesis is a very active area of research, and many high level design tools and approaches exist (e.g. ImpulseC, AutoESL, SysGen, catapultC). Primarily, the focus of these tools is to reduce time to solution for designers who have considerable computer engineering expertise. While working from a high level of abstraction and guaranteeing functionally correct output, they require design input (e.g. pragma statements) concerning low-level hardware detail in order to produce efficient RTL output. Hence, such languages are generally inappropriate for use by those in the life-sciences community, which is comprised primarily of software programmers with limited hardware design experience.

A. Convey personality design flow

The use of heterogeneous computing architectures like the Convey HC platform (CPU + FPGA) for fixed-point algorithms [8] or Nebula (CPU + GPU) for floating point [9], constitutes one approach to meeting such big-data processing demands. Convey Computer's Hybrid-Core family of platforms are examples of such architectures. The first generation of these Hybrid-Core servers, the HC-1 and HC-1ex, are considered in this work.

The Convey HC-1 system consists of a commodity Intel Xeon host server extended with a custom co-processor board. This co-processor board consists primarily of four large FPGAs (Xilinx part XC5VLX330) called Application Engines (AE), augmented with cache-coherent, high bandwidth memory access (8 memory controllers per AE). Each configuration of the Convey co-processor is called a "personality," and provides application-specific functions available to the host server processor as custom instructions [10].

Convey's Personality Development Kit (PDK), a flow for the development of custom personalities, is illustrated in Figure 2. The first step of this development process is to identify the target application's computational kernel and performance bottlenecks, analyzing heavily-used data structures and algorithm parallelism. Options for implementing the application in hardware are then evaluated. This requires knowledge of the hardware architecture and the FPGA resources available to the personality. Once an implementation strategy is established, the functions performed by the hardware design can then be mapped to the set of instructions in the Convey Instruction Set Architecture reserved for custom personalities. The customs instructions available to a PDK personality are defined in [11].

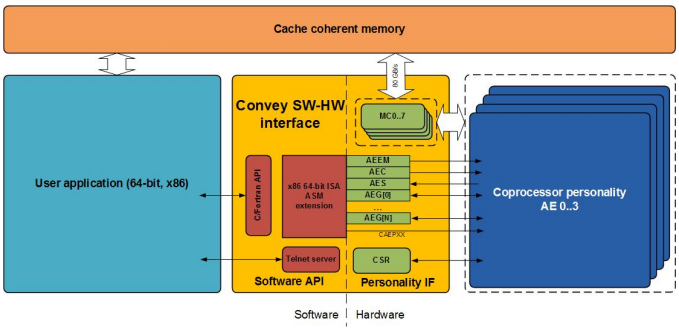

Fig. 3. Overview of the Convey hardware-software interface.

Prior to actual hardware implementation, a software model emulating the desired co-processor function is developed for the purpose of verifying the hardware kernel. Convey provides an simulation environment which allows rapid prototyping of both the hardware and software components of a custom personality. The provided simulation environment includes Verilog models of AE instruction dispatch, register states, and the memory subsystem. The software model is then simulated along with the rest of the system to prove and debug the kernel architecture. Once the personality software model is in place, HDL development is begun. The accelerator functionality is specified in HDL (Verilog or VHDL) or synthesized netlists (e.g. EDIF). the implementation flow results in a compressed personality package. Convey provides a hardware simulation environment with bus-functional models of all AE interfaces, in order to support functional verification of the accelerator HDL specification. The architecture simulator is used to provide stimulus to the HDL simulation through a Verilog Procedural Interface (VPI). Finally, the accelerator functionality is implemented by the Xilinx FPGA design flow and tested on the Convey co-processor hardware. The Personality Development Kit provides Verilog user logic wrappers that facilitate memory access and communication between the host application and the kernel accelerator.

Figure 3 displays the architecture of the Convey hardware-software interface, which joins the host software application with the co-processor personality, which executes in parallel on up to four AE FPGAs. This architecture provides significant computational capability; however, the PDK does not address the programming needs of big-data users, who typically do not have knowledge of HDL system descriptions nor experience in digital systems design and FPGA implementation flows.

B. Azido system description

Azido (Figure 1) is a graphical, object-oriented design environment based on the Implementation Independent Algorithm Description Language (I2ADL). The tool attempts to abstract the low-level complexities of digital hardware design to a high level more intuitive to a user without hardware design experience. Furthermore, the tool simplifies and accelerates hardware design by encouraging object reuse, and providing an extensible core library of implementation-independent algorithmic objects, known as the *CoreLib*. Bitstream implementation of these algorithms is accomplished through the use of Azido System Descriptions–behind-the-scenes, script-based implementation "plugins" that both describe the target

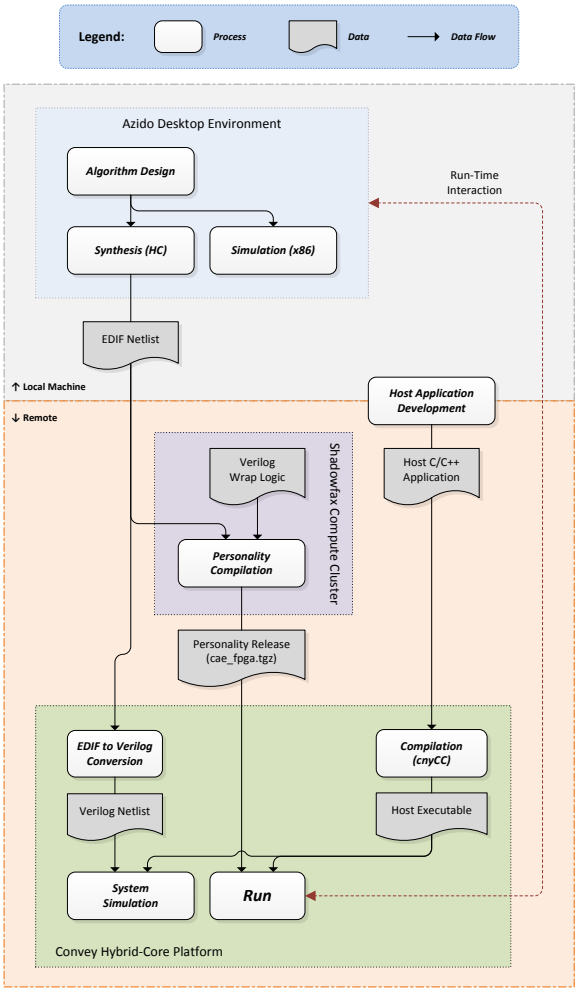

Fig. 4. Movement of data within bFlow.

system architecture and consume a synthesized logical netlist generated by Azido, producing a configuration for a specific platform.

Among several existing graphical design environments, Azido was selected as a front-end for three primary reasons: 1) it provides a very flexible design environment and core library capable of servicing many application domains at both low and high levels of abstraction, 2) the System Description-based implementation framework creates opportunities for easily extending the tool to support many target platforms, and 3) in addition to standard schematic-capture abilities, Azido provides some dynamic features, including automatic data type conversion and graphical polymorphism. These characteristics distinguish Azido from tools such as LabVIEW [12] and Simulink [13]. The former is similar in terms of the usability of the design environment, but is constrained to specific target platforms. The latter, though capable of producing platform independent HDL, lacks most of the dynamic features mentioned above.

C. bFlow Contributions

The bFlow approach provides a simplified and portable accelerator development flow which supports rapid prototyping

of big-data algorithms in hardware. Accelerator design productivity is improved by using an intuitive, graphical front-end (Azido) and hiding low-level details of the hardware implementation (e.g. provision of memory stream abstraction). A reduction in compilation time of the accelerator is achieved by employing incremental implementation strategies facilitated by qFlow [14] and Xilinx Hierarchical Design Flow [15]. Furthermore, because of the graphical front-end and abstract objects, the flow is better-suited for designers without HDL expertise.

This framework supports the growth of the third-party "personalities ecosystem" through the contribution of custom accelerator implementations, in a way similar to that observed in the Linux community [16]. Thus, no user is limited to the closed-source personalities provided by the system vendor. Finally, the bFlow approach is portable–it supports local design and remote accelerator implementation using an HPC cluster (e.g. VBI's Shadowfax compute cluster [17]), as well as remote execution on the Convey Hybrid-Core platforms. The designer need only install Azido and the Convey HC System Description on his local machine.

III. Rapid Bio-accelerator development flow

The approach is divided into two efforts–the assembly of a usable front-end design environment and the development of a fast compilation back-end that increases productivity. The movement of data within the final flow, at a high level, is shown in Figure 4. Algorithm design is performed within Azido on the user's local machine. Upon completion of synthesis, which is performed by Azido, an EDIF netlist is transferred to VBI's Shadowfax cluster [17], where it is combined with custom wrap logic and implemented to a bitstream using an accelerated compilation process or Convey's standard flow. The generated personality is packaged and placed in shared storage accessible to VBI's Convey HC servers. Given the software for the host CPU compiled with the Convey C/C++ compilers, the design can be executed on the HC platform.

Local testing of individual logical blocks is accomplished by Azido's built-in x86 system description, and rough system simulation can be done locally by instantiating Component Object Model (COM) objects throughout the design. For example, file streaming COM objects can be used to approximate the functionality of the memory stream objects in the HC system description. Alternatively, cycle-accurate simulation of the entire HC system (host CPU + FPGA application engines) can be performed on the server itself using Convey's supported simulation flow and a structural Verilog netlist of the Azido design, generated by an EDIF-to-Verilog conversion utility developed in-house.

A. Convey HC-1 system description and software helper routines

The assembly of a design environment usable by those without digital design expertise was accomplished through the use of Azido with a Convey HC-1 system description and a collection of convenient software routines. The system description can be divided into three distinct components: 1) a communication implementer for the transfer of diagnostic probing and user stimulation between Azido and the personality at run-time, 2) an abstracted view of the co-processor dispatch interface, and 3) streaming abstractions for the memory interface. Compilation of the Azido design produces a Hybrid-Core personality, the architecture of which is shown in Figure 5.

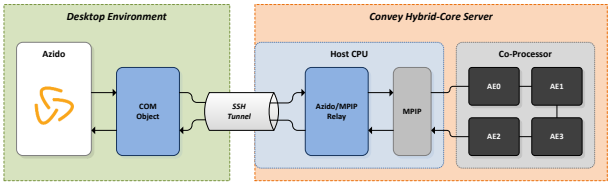

Fig. 6. Azido/Hybrid-Core communication architecture.

1) Azido communication implementer: The first component is invisible to the designer, and consists of communication of data between the Azido environment and the personality for the purpose of run-time diagnostic probing and stimulation of the design. This is achieved through the use of a COM object instantiated in the Azido System Description, an SSH tunnel from the designer's desktop to the HC server, a utility running on the Convey x86 host processor, and a Convey-provided telnet server, mpip, which provides access to the HC's management ring interface. Azido transmits user input from the run-time widget window to the COM object, which in turn transfers the data through the tunnel to the relay utility on the Convey platform host processor. This process sends register read/write commands through the mpip server to the management ring, both capturing and exciting signals in the design (see Figure 6).

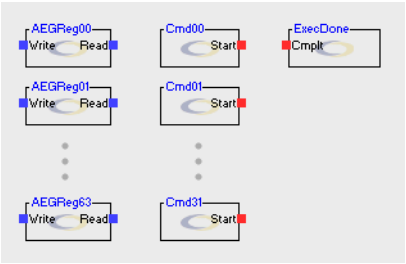

Fig. 7. Collection of Convey dispatch interface abstractions in Azido.

2) Convey dispatch interface abstraction: The second consists of the Azido abstraction of the AE dispatch interface (Figure 7). Logic external to the Azido-generated netlist decodes incoming dispatch instructions and presents a simplified interface to the Azido designer. The AE general (AEG) registers are exposed as simple read/write blocks in Azido, and the co-processor custom instructions are exposed as blocks with a "Start" output. Management of the co-processor idle state is implemented external to the Azido design.

3) Memory streaming abstraction: Lastly, to conceal the complexities of random memory access and address arithmetic, bFlow contains a simplified, streaming abstraction to the memory controllers available to each AE. These "streamers" present two abstractions to the designer: (1) a "source" module and (2) a "sink" module, to be used to stream a block of data from memory into the AE and stream data out to memory, respectively. Figure 8 provides a usage example of the Azido

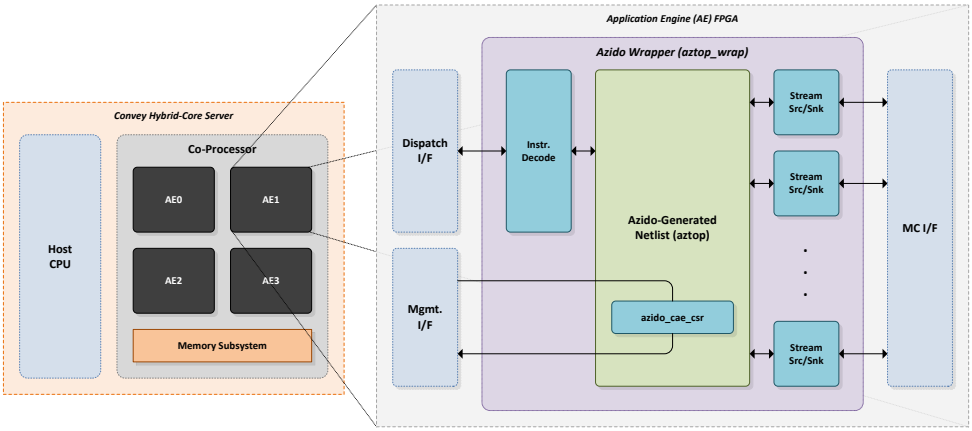

Fig. 5. The bFlow personality application engine (AE) architecture.

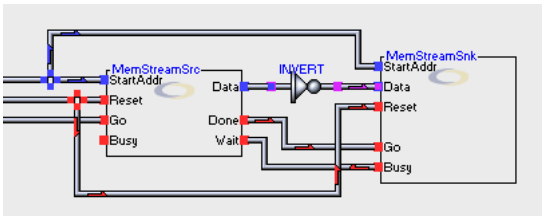

Fig. 8. Usage example of the memory stream sink and source objects in Azido. This algorithm bitwise inverts a word of memory each time the Go input of the *MemStreamSrc* object is asserted.

modules for these two abstractions, both of which comply with Azido's GDBW flow-control convention.

In addition to the System Description objects available to the designer within Azido, several software routines were developed to simplify the configuration and execution of a custom personality. This is accomplished by wrapping the Convey-provided, low-level assembly routines in higher-level, C/C++ routines contained in a C++ class. These routines include helper functions for reading and writing the AEG registers, calling co-processor custom instructions, and copying data to/from co-processor memory. Furthermore, a subclass containing additional abstractions that simplify interaction with the Azido streaming objects is also provided.

B. Extension of Smith-Waterman reference implementation by non-CS/CE users

To verify the usability and productivity that bFlow supports, it was placed in the hands of four non-engineering participants of the NSF-funded Summer Institute organized by VBI. Prior to the start of the event, the students were asked to review [18] and gain a basic understanding of the systolic array approach to implementing the Smith-Waterman sequence alignment algorithm. The students were given a reference implementation of the Smith-Waterman matrix fill operation, and tasked with accelerating the design and extending its functionality.

1) Simple systolic array, wavefront computation of scoring matrix: The Smith-Waterman algorithm [19], a local sequence alignment algorithm proposed in 1981 and well known to the bioinformatics community, is often used to compute the similarity between two nucleotide or protein sequences, S and T, by computing a scoring matrix of size $S \times T$. This work considers a common case in which S, the query, is quite short–at most one or two thousand bases long, and T, the reference, is quite long–millions of bases. This algorithm yields the optimal local alignment, given a character substitution matrix and affine gap penalties. The algorithm consists of two steps: 1) computation of the scoring matrix and location of best alignment and 2) the traceback of the best alignment starting from the matrix cell with the highest score. Since performing the traceback operation requires only the subset of the scoring matrix containing the best alignment, this step is often postponed until the location of the best alignment is determined, which can be done by tracking the highest-scoring cell during the first step.

A reference implementation (Figure 9) based on [18] was created in the Azido environment and given to the participants along with the task of adding functionality and improving performance. The students were given approximately one week to modify the design, and restricted to only two memory streamer objects.

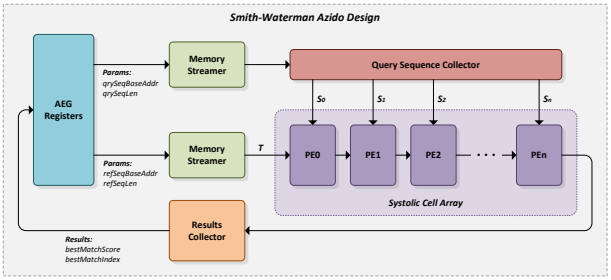

Fig. 9. High-level architecture of the reference systolic array implementation of the Smith-Waterman scoring matrix fill operation. This was given to the NSFSI participants with the task of adding functionality and improving throughput within about a week.

C. Partial implementation flows with qFlow and Partitions

Acceleration of personality compilation is achieved by two methods, both partial implementation flows. Both are *frameworks*, accomplishing improved build times through high-level management of the implementation process, making use of Xilinx ISE for core implementation functionality (synthesis,

placement, routing, etc.). The first is an application of the Xilinx Hierarchical Design Partitions flow [15], while the second involves the use of qFlow, a back-end compilation framework [14]. Both methods exploit the high-level architecture of all Convey personalities–specifically, the inclusion of interface logic that remains nearly static throughout the the development process–it is configured once and then left alone. This logic consists mostly of interfaces to the eight available memory controllers and a memory crossbar, as well as logic that communicates with the dispatch and management processors. Also included is the wrapper user logic surrounding the Azido-generated netlist (see Figure 5), which is responsible for instruction decode and abstraction of the provided memory interface into a simple, streaming interface. This "static" logic consumes roughly 25% of each of the HC-1's AEs (Xilinx part XC5VLX330) and, when using Convey's traditional compilation process, is re-implemented each build. Given that this portion of the design changes Both of the following partial implementation concepts accelerate compilation by implementing this logic once, and then preserving its placement and routing during consecutive builds of the personality.

1) Partitions flow: The first approach taken makes use of the Xilinx Hierarchical Design Partitions flow [15]. Two partitions were selected for this flow: (1) top partition containing the entire FPGA design and (2) the Azido-generated logic. The first is preserved while the second is constantly updated by the Azido designer. The implementation of this flow is very simple, consisting of some Makefile extensions and a couple constraint files (*.ucf).

2) qFlow: This second utilizes a subset of the qFlow framework, a tool for acceleration back-end compilation. The tool assumes a hierarchical design methodology similar to that which the Convey PDK encourages. That is, the use of interface logic that is designed and configured once, experiencing few evolutions throughout the entire design process, during which the core, computational logic that is the focus of the design undergoes many evolutions. Compared to the partitions-based approach discussed above, qFlow provided generally faster compilations (see Section IV-A), but was unable to fit some of the larger designs that the partitions framework was able to.

IV. Results

The participants were successful in their attempt to improve the performance and functionality of the reference S-W implementation. Modifications included logic to maintain the index of the highest scoring alignment, and a transition from a single cell array to multiple arrays, with the goal of achieving a linear speedup proportional to the number of arrays by searching multiple partitions of the reference sequence in parallel. The first modification was simple, and included the use of Azido's counter, maximum, multiplexer, and register objects; however, the second involved significant change to the high-level structure of the implementation–specifically, duplication of the systolic cell array and the addition of logic at the front and back of the arrays to split the incoming stream and collect results from each array, respectively. The improvements resulted in a realized 4x bandwidth increase from 150 million to 600 million bases per second, with a feasible speedup of

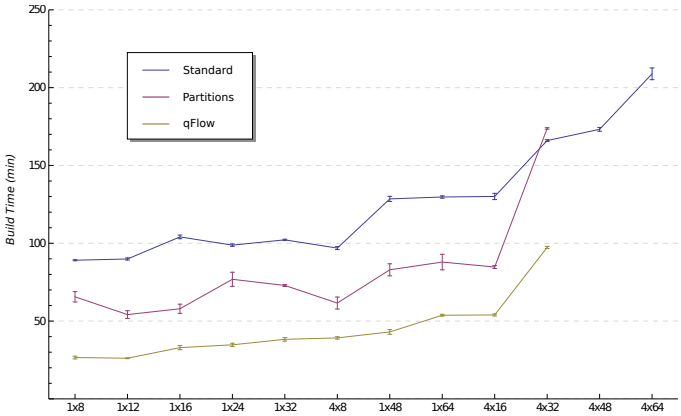

Fig. 10. Build times for Convey's standard flow, the Partitions-based flow, and qFlow for twelve different variations of the Smith-Waterman Azido implementation.

32x to 4.8 billion bases per second, given enough parallel cell arrays (note that, as the parallel array count increases, the supported query sequence length decreases). These reported throughput rates are for co-processor performance only, and assume that the reference sequence(s) are available in co-processor memory.

3) Usability challenges encountered by the group: One difficulty experienced by the students during their use of the Azido environment was gaining a basic knowledge of synchronization–especially, synchronizing multiple flows of data. Azido's CoreLib, a core library of modules available to the user, makes use of the Go-Done-Busy-Wait protocol to simplify synchronization. Multiple, out-of-sync data flows can be joined through the use of special "SyncQueuePair" objects. However, most of these GDBW-based modules use registers or queues for flow control, and the excessive use of such modules may result in inefficient utilization of FPGA resources.

Another complication of the design process was timing closure. Under the standard parameterization, the Convey PDK enforces a clock rate of 150 MHz. Such a tight constraint is easily broken by long chains of asynchronous operations. However, Azido neither analyses the design nor enforces any timing restrictions at compile time. Hence, whether or not a design meets timing is determined only during the "behind-the-scenes" implementation process, and the abstraction of the implementation that Azido provides is lost if a constraint is not met and the design proves dysfunctional.

A. Compilation performance

The performance of the alternative build flows is given in Table I and visualized in Figure 10. Each Smith-Waterman configuration is named with convention sw_MxN, where M is the number of parallel systolic arrays and N is the length of each array. The median speedup over the standard, Convey-provided flow for the partitions-based approach 1.51, while that of qFlow was 2.76. The last two configurations tested, 4×48 and 4×64, could not be placed into the dynamic region–the same dynamic region constraints were used for both flows. Note the jump in build time from configuration 4×16 to 4×32 due to the increased utilization of the dynamic region.

TABLE I. BUILD TIMES (AVERAGE OF THREE RUNS) FOR CONVEY'S STANDARD FLOW, THE PARTITIONS FLOW, AND QFLOW. FOR THE PARTITION AND QFLOW RESULTS, THE SPEEDUP IS GIVEN IN PARENTHESES. THE DEVICE UTILIZATION REPRESENTS THE UTILIZATION DUE ONLY TO THE DYNAMIC, USER LOGIC, RATHER THAN THE ENTIRE DESIGN (THIS CAN BE ADDED TO THE UTILIZATION DUE TO THE STATIC LOGIC TO GET TOTAL DEVICE UTILIZATION).

Design	# Cells	Device Utilization (%)		Mean Build Time (min)		
		Slice LUTs	Slice FFs	Standard	Partitions	qFlow
sw_1x8	8	1.83	2.16	89.10	65.60 (1.36)	26.58 (3.35)
sw_1x12	12	2.43	2.44	89.90	54.16 (1.66)	26.12 (3.44)
sw_1x16	16	3.02	2.73	104.09	57.92 (1.80)	32.94 (3.16)
sw_1x24	24	4.22	3.30	98.80	76.85 (1.29)	34.75 (2.84)
sw_1x32	32	5.41	3.87	102.20	72.89 (1.40)	38.27 (2.67)
sw_4x8	32	5.62	4.10	96.90	61.58 (1.57)	39.20 (2.47)
sw_1x48	48	7.79	5.01	128.50	82.95 (1.55)	43.02 (2.99)
sw_1x64	64	10.18	6.15	129.73	87.92 (1.48)	53.75 (2.41)
sw_4x16	64	10.36	6.34	130.08	84.74 (1.54)	53.94 (2.41)
sw_4x32	128	19.85	10.81	165.99	173.62 (0.96)	97.33 (1.71)
sw_4x48	192	29.34	15.29	173.22		
sw_4x64	256	38.83	19.76	208.89		

V. CONCLUSIONS AND FUTURE WORK

This work presents bFlow, an FPGA-based big-data accelerator development environment significantly more usable by non-engineers than traditional accelerator design tools. This is accomplished by utilizing a graphical front-end environment and a seamless, accelerated compilation back-end. Productivity is improved by encouraging object reuse, providing high-level design abstractions, and increasing design turns-per-day by reducing compile time.

The framework was applied to the Convey Hybrid-Core heterogeneous computing platforms, and its usability and productivity tested by participants of the NSF-funded bioinformatics Summer Institute at the Virginia Bioinformatics Institute. Within a week, the participants successfully extended and accelerated a reference Smith-Waterman FPGA implementation designed in Azido, achieving a 4x throughput increase and additional functionality. However, significant deficiencies in Azido's graphical syntax, specifically the poor synchronization abstractions, were uncovered immediately and proved to be a major barrier to the participants' creation of complex logic.

In addition to the front-end developments, compilation times for the Convey HC-1 server were reduced through the use of the Xilinx Partitions flow [15] and the qFlow framework [14] as alternatives to Convey's standard approach. The median compilation speedups for these flows were 1.51 and 2.76, respectively.

Future research efforts include the consideration of memory abstractions other than simple stream producers and consumers. Also, while bFlow currently provides a small software library to simplify the hardware-software barrier on the Convey HC platform, there is great potential in describing both the hardware and software together within Azido. Lastly, the consideration of other design entry tools, such as LabVIEW [12], is a must.

ACKNOWLEDGMENT

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant Nos. EEC-0642422 and IIP-1161022, and by NSF Award No. OCI-1124123, High Performance Computing in the Life/Medical Sciences.

REFERENCES

[1] E. D. Feigelson and G. J. Babu, "Big data in astronomy," *Significance*, vol. 9, no. 4, pp. 22–25, 2012. [Online]. Available: http://dx.doi.org/10.1111/j.1740-9713.2012.00587.x

[2] K. Cukier, "Data, data everywhere," *The Economist (London)*, vol. 394, no. 8671, p. 3, 2010.

[3] G. Brumfiel, "Down the petabyte highway," *Nature (London)*, vol. 469, no. 20, p. 282, 2011.

[4] L. J. McIver, J. W. Fondon III, M. A. Skinner, and H. R. Garner, "Evaluation of microsatellite variation in the 1000 genomes project pilot studies is indicative of the quality and utility of the raw data and alignments," *Genomics*, vol. 97, no. 4, pp. 193–199, 4 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0888754311000024

[5] M. S. Rosenberg, *Sequence alignment : methods, models, concepts, and strategies*. Berkeley: University of California Press, 2009.

[6] Convey Computer, "The convey hc-1 computer architecture overview," http://conveycomputer.com/Resources/ConveyArchitectureWhiteP.pdf.

[7] B. Nelson, M. Wirthlin, B. Hutchings, P. Athanas, and S. Bohner, "Design productivity for configurable computing," in *ERSA '08: Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, 2008, pp. 57–66.

[8] K. Pereira, P. Athanas, H. Lin, and W. Feng, "Spectral method characterization on fpga and gpu accelerators," *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, pp. 487–492, Nov. 30 2011-Dec. 2 2011.

[9] W. chun Feng and K. W. Cameron, "The green500 list: Encouraging sustainable supercomputing," *Computer*, vol. 40, no. 12, pp. 50–55, Dec. 2007.

[10] J. D. Bakos, "High-performance heterogeneous computing with the convey hc-1," *Computing in Science & Engineering*, vol. 12, no. 6, pp. 80–87, Nov.-Dec. 2010.

[11] Convey Computer, "Personality development kit (pdk) for convey hybrid-core computers," http://conveycomputer.com/Resources/PersonalityDevelopmentKit.pdf.

[12] National Instruments, "Ni labview - improving the productivity of engineers and scientists - national instruments," http://www.ni.com/labview/, 2012.

[13] MathWorks, Inc., "Simulink - simulation and model-based design," http://www.mathworks.com/products/simulink/, 2012.

[14] T. Frangieh and P. Athanas, "A design assembly framework for fpga back-end acceleration," in *International Conference on Reconfigurable Computing and FPGAs (ReConFig 2012)*, to appear.

[15] Xilinx, "Hierarchical design methodology guide," http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/Hierarchical_Design_Methodology_Guide.pdf.

[16] E. Raymond, "The cathedral and the bazaar," *Knowledge in society*, vol. 12, no. 3, pp. 23–49, 1999.

[17] Virginia Bioinformatics Institute, "Partnership supercomputing program," https://www.vbi.vt.edu/high_performance_computing/.

[18] P. Zhang, "Implementation of the smith-waterman algorithm on a reconfigurable supercomputing platform," *Proceedings of the 1st international workshop on High-performance reconfigurable computing technology and applications held in conjunction with SC07 - HPRCTA '07*, p. 39, 2007.

[19] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences." *J Mol Biol*, vol. 147, no. 1, pp. 195–197, Mar 1981.