Applying Field Programmable Logic Arrays to Biological Problems

Daryl Popig[1], Debra Ryle[1], David Pellerin[2] and Eric Stahlberg[3]

[1]Accelerated Data Concepts, LLC Columbus, OH
Email: {dpopig, dryle}@acceleratedata.com
[2]Impulse Accelerated Technologies, Inc. Kirkland, WA
Email: david.pellerin@ImpulseC.com
[3] Ohio Supercomputer Center, Columbus, OH
Email: eas@osc.edu

Abstract

Today's vast amount of bioinformatics data is growing beyond the researcher's ability to analyze and derive value from the information in a reasonable amount of time. Parallel software acceleration technologies are increasingly critical for achieving the necessary processing rate for this information. In this paper, we will show how one parallel technology, a Field Programmable Gate Array (FPGA), can be applied in biological problem solving computer strategies to increase computational speed.

Background

FPGA (Field Programmable Gate Array) technology uses large-scale integrated circuits made up of programmable logic that can evaluate an immense array of the basic logic gate functionality (AND/OR/XOR/NOT) comprising complex combinatorial functionality such as decoders and adders. This combined logic can be easily programmed and reprogrammed to the set of interconnected hardware-level gates to specific settings which implement a computationally extensive algorithm. Because FPGAs are inherently parallel devices, the FPGA can take advantage of data level parallelism by splitting the data across multiple FPGA internal pipelines to attain the required speedup. Several bottlenecks associated with general-purpose processors are eliminated given that FPGAs can perform parallel operations concurrently on large streams of data.

Biological applications that require computational complexity such as pattern matching, multiple sequence alignment, structural genomics, protein-ligand binding for drug discovery, network analysis, and gene sequence analysis can take advantage of the highly parallel execution provided by using FPGAs. Application speedup can impact research efforts by reducing the time it takes to analyze the data. This paper will give a brief outline of how FPGAs can be applied to computational biological problems and what FPGA tools are currently available to researchers in the life sciences field.

Computational Challenges in Research

Using traditional parallel software engineering techniques, research software must employ large numbers of general purpose processors (e.g. clusters) to complete complex operations within a prescribed timeframe. With the general

leveling off of processor clock speeds and the corresponding end of Moore's law as it applies to processor capabilities, new approaches are needed to reach the next level of computational performance. FPGA technology has reached a state where performance gains in this technology are anticipated to exceed gains in general-purpose processors [Underwood and Hemmert, 2004].

Many bioinformatics applications have both shown and delivered on the promise of using FPGA technology to accelerate performance relative to a general purpose processor [Smith and Stahlberg 2006; Fernando et al 2006; Regester et al 2005; Bondhugula et al 2006]. The most prevalent successful implementation is the Smith-Waterman algorithm used to identify similarities among bioinformatics sequences. Speed-ups on the order of 30-fold have been reported for this algorithm [Regester et al 2005]. Graph network analysis implementations of the classic Floyd-Warshall algorithm have shown similar speedups [Bondhugula et al 2006].

Another such example, a drug docking application that simulates the molecular interactions of a drug candidate in-silico, can benefit from highly parallel execution. The code, written in C, will run the simulation of the atomic interactions. Similar to molecular dynamics simulations, [Smith et al 2005] these docking simulation runs require algorithms to calculate the interaction energies of all pair-wise atom interactions. This now becomes a massively parallel problem that is ideally suited for the FPGA. The FPGA can be custom loaded with efficient data structures and algorithms specific to the atom to atom calculations that are required.

Another challenge for scientific applications is the need to search through unstructured and semi-structured data. Much of this unstructured data is in the form of digital images. Efficiently searching through unstructured data (video, audio, instrument signals, etc.) requires special programming beyond that available in the standard SQL specification. FPGA accelerated data searches have been shown to gain as much as a 40% increased speedup.

Applying FPGAs to Meet these Challenges

An FPGA is a semiconductor IC (Integrated Circuit) device containing programmable components that can be connected together to form digital logic circuits made up of basic gates, counters, adders, shift registers, flip-flops, memory and many other combinational, composite or sequential logic elements. These fundamental circuits can be interconnected to form entire digital systems on a programmable chip.

FPGAs can increase the computational speed in scientific/biological calculations by acting as co-processors allowing conventional microprocessors (CPUs) to offload computationally expensive data processing. During system initialization, the FPGA is programmed with a bit stream representing the hardware accelerator. This bit stream is used to configure the logic gates inside the FPGA, and implements a specific algorithm in hardware. Once the FPGA has been configured, the CPU can then send the

data to be processed to the FPGA's hardware-implemented algorithm, where the data is processed and the result is returned to the CPU. When a different algorithm is required, the CPU simply reloads the FPGA with the appropriate bit stream which codes for the new algorithm and the device is ready to do an entirely different calculation using hardware.

Emerging Approaches to FPGA Application Development

Creating applications that involve FPGA acceleration requires different design methods and tools than traditional software design. Scientific computation advantages can best be realized by providing a seamless integration path between the software application and the FPGA. Using this approach, a domain scientist can focus on the science and research process rather than on the software/hardware interfacing and FPGA tool development. Fortunately, due to advances in FPGA compiler tools, the bit stream that codes the algorithmic computation can be programmed by the scientist/user using standard C programming techniques.

Using standard C for application development has many advantages, not the least of which is the opportunity to use iterative, software-oriented methods of design optimization and debugging. With the Impulse C tools, for example, both hardware and software elements of the complete application can be described, partitioned and debugged using standard C programming tools such as GCC and GDB. During this process, then application programmer can make use of familiar C-code optimizations to increase performance, without having a great deal of FPGA-specific hardware knowledge.

The resulting optimized C code is compiled by the FPGA development tools (in particular, the C-to-hardware compiler) to create a parallel hardware/software implementation. The FPGA map, place and route tools then take the output from the compiler—that portion of the application representing the application accelerator(s)—and convert that into an FPGA bit stream. The bit stream is then loaded into the FPGA before computation is started. The complete C-to-hardware compilation process can be summarized as:

- Describe the complete application in C language and use a standard C debugger to verify the algorithm.
- Profile the application to find the computational "hot spots".
- Use data streaming, message passing and/or shared memory to partition the algorithm into multiple communicating software and hardware processes.
- Use interactive optimization tools to analyze and improve the performance of hardware-accelerated functions.
- Use a C-to-hardware compiler to generate synthesizable hardware, in the form of hardware description language files.
- Export the generated hardware description files to the FPGA optimization and mapping tools.

- Download the resulting FPGA bitmap to the target FPGA device.

The only drawback to the above approach is that, because the FPGA circuitry is so dense, the FPGA mapping tools can require a considerable amount of compile time to finish. This, however, has no effect on the application run time because, once generated, the FPGA bit file can be stored on disk (or in some cases in Flash RAM) and reloaded as needed.

On the hardware side, the FPGA is usually mounted to a PCI board that can be mounted into a high performance desktop or work station. Accelerator board examples include the Nallatech BEN series of PCI boards and the Annapolis Micro Systems Wildforce series of PCI boards. Larger systems such as the Cray XD1, SRC Computers SRC-7 or the SGI RASC can have multiple nodes with multiple FPGAs and multiple CPUs. These high performance hybrid parallel systems can take advantage of both the conventional microprocessors and the reconfigurable FPGA hardware resources to do the most efficient computing and partitioning of the hardware and software portions of the code to achieve maximum speedup.

Ultimately, the system chosen should be the one that best suits the researcher's needs. By looking at the ease of development of the tool sets, technical support available, the necessary data bandwidth, scalability, power consumption, price, communication among the nodes, number of FPGAs, number of microprocessors, clock speed, and the amount of memory, an appropriate system can be characterized and procured.

On the software side, there are two primary challenges to address to be successful in FPGA application development. The first challenge to a new FPGA user is to define which algorithms can readily exploit FPGA fine-grained parallelism for the problem. These algorithms have historically proven to be those with low-level data parallelism and an absence of floating point operations. More recently, floating point libraries have become available that promise to increase the range of candidate algorithms.

Once a candidate algorithm has been identified, a second closely related challenge emerges in algorithm partitioning. In order to be successful with current hybrid architectures, the algorithm must generally be partitioned in a manner which minimizes the communication to the FPGA while maximizing the actual computation on the FPGA.

Our experience with using a single FPGA on the Cray XD1 for a searching project proved that the concept of noninvasive interfacing an FPGA to an enterprise database system is entirely possible [Ryle et al 2005]. We were able to demonstrate a seamless integration strategy with a popular database environment (Oracle 10G on Solaris) and a programmable FPGA server (Cray XD1). While using resource management frameworks such as the Sun Grid Engine, we wanted to demonstrate the feasibility of hardware resource sharing without first deploying more complicated grid and resource management infrastructures. In both deployment architectures, we were successful at application integration with a remote FPGA. This further

demonstrates the ease in which an FPGA server can be integrated into a production database environment, with minimal disruption, for complex query search, development and optimization.

The authors also have experience in implementing the Smith-Waterman algorithm on an FPGA and have seen publications boasting computational speedups of 64-fold over a conventional CPU implementation on the Cray XD1 [Margerm 2006]. The Smith-Waterman algorithm is used in bioinformatics and uses dynamic programming concepts that will take an alignment of any length, and at any location, in any sequence, and determines whether there is an optimal alignment that can be found.

Emerging Tools for Rapid FPGA Development

The design approach starts with an analysis of the problem to determine which part of the problem is best suited to execution in the FPGA. The program should be partitioned into software (executed by general-processors) and hardware (executed in the FPGA). A likely candidate to load in hardware would be any part of the program that can be run in separate parallel threads or any routines that contain computational routines. Keep in mind that data movement between these two partitions should be considered when deciding the partition divisions. In fact, using an FPGA for acceleration of certain algorithms can result in a simple shifting of the processing bottleneck from one of computation, to one of data movement. It is therefore important to make use of higher-level tools that support fast prototyping and iterative analysis.

Programming tools to assist in FPGA application development are emerging from companies such as SRC, Impulse, Mitrionics, Nallatech, and DSPlogic. These tools allow programs written in C (or in C-like languages) to be converted into logic appropriate for FPGA mapping. A complete tool flow, including software provided by the FPGA manufacturer, takes the C code through the map, route and place steps while creating an optimal FPGA routing.

To allow increased use of statement-level and system-level parallelism, these tools are supplied with functions (Impulse C), macros (Mitrionics) or directives (SRC) that can be referenced in the software code to interface with the hardware code, for example to set up and manage a streaming or shared memory interface. This is important because the design of software-to-hardware communications may have as great an impact on application performance as actual computation acceleration.

Once the software code is compiled on general-processors and the hardware converted to bin modules and loaded into the FPGA, the two components—hardware and software—interact through the generated interfaces.

An international effort has started as a result of the diversity in approaches to FPGA application development. The effort, called OpenFPGA (see www.openfpga.org), is a cooperative effort among FPGA application users, FPGA manufacturers, and FPGA software vendors to ultimately standardize elements of the FPGA application to improve overall portability. At the present time, the effort has participation from over 300

individuals worldwide in over 30 countries [Stahlberg et al 2006]. In the near future, common approaches to FPGA application development are expected to emerge, easing further the effort required to create high-performing FPGA-enhanced applications.

Summary

In summary, we have shown that life science algorithms can take advantage of FPGAs as a viable resource to improve highly computationally expensive processing by moving expensive computations from the CPU and into the specifically designed logic inside the FPGA. Algorithms such as the Smith-Waterman have shown significant speed improvement when implemented on a FPGA while integration to current infrastructure such as databases is possible. FPGAs are becoming easier to use as the development tools get better and as the prices on FPGAs falls as smaller/denser chip manufacturing technology becomes available, thus making them affordable to use in more computing applications. Support is also increasing as FPGAs now have the attention of world-wide computer scientist and developers though OpenFPGA where standards are being derived to open up FPGA use for a wide range of high level applications.

References

U. Bondhugula, A. Devulapalli, J.Dinan, J. Fernando, P. Wyckoff, E. Stahlberg, P. Sadayappan, "Hardware/Software Codesign for All-Pairs Shortest-Paths on a Reconfigurable Supercomputer", 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, (FCCM '06), 2006

S. Margerm, " Reconfigurable Computing in Real-World Applications", FPGA Journal, February, 2006.

K. Regester, J. Byun, A. Mukherjee, A. Ravindran, "Implementing Bioinformatics Algorithms on Nallatech-Configurable Multi-FPGA Systems", XCell Journal, Second Quarter, 2005

D. Ryle, D. Popig, E. Stahlberg, "Integrating FPGA Technology and Database Management Systems on Solaris Systems", Proceedings Sun Users Performance Group Conference, Arlington, VA, April 18, 2005

H. Smith and E. Stahlberg, "XD1 Implementation of a SMART Coprocessor for Fuzzy match in Bioinformatics Applications", in process

M. Smith, J. Vetter, X. Liang, "Accelerating Scientific Applications with the SRC-6 Reconfigurable Computer: Methodologies and Analysis", Proceedings Reconfigurable Architectures Workshop (RAW), 2005.

E. Stahlberg, K. Wohlever, D. Strenski, "Status Report of the OpenFPGA Initiative: Efforts in FPGA Application Standardization", in process.

E. Stahlberg, J. Fernando, J. Doak, K. Wohlever, "Accelerated Biological Meta-Data Generation and Indexing on the Cray XD1", Proceedings Cray Users Group Conference, Albuquerque, NM, May 15, 2005