Chapter 2
Related Work

Abstract In this chapter we discuss the related work, where we present some of the ideas and implementations reported by other researchers working in areas closely related to this field. First we describe the various accelerator architectures and then, discuss FPGA based accelerators. We describe the FPGA architecture as well as the EDA tool flow followed while exploring HEBs in FPGAs. We discuss "bioinformatics" domain and the two important applications belonging to this domain. We show how these applications have benefited by FPGA-based acceleration.

2.1 Accelerator Architectures

Accelerators have become very popular in the community in the past decade. Popular implementations are based on ASICs, FPGAs, GPUs, and CELL add-ons. Heterogeneous accelerators consist of a mix of accelerators which are connected to the control processor. Each of the approaches listed here have their own strengths and weaknesses.

Chung et al. [22] study the importance of heterogeneous architectures. They evaluate different architectures by running various benchmarks and predict the performance of future architectures using scaling factors [43]. They predict that the future systems will include a mix of various architectural features in order to have a balance between power and speed. Venkatesh et al. [85] predict that the future chips will have large amounts of "Dark Silicon". As the operating frequency of IC is increased, the chip gets heated up and some parts of the IC have to be shut down to avoid permanent damage. The parts which are shut-down are known as dark silicon. The authors propose reconfigurable "conservation cores" that operate at lower frequencies during the shut-down of the faster cores. Even though these conservation cores have less operating frequency, they carry out specialized computations parallel and thus provide performance benefits over chips that do not have them.

In ASICs, hardware implementation of multiple copies of the compute intensive kernels is done. ASIC implementation of the kernels will give higher performance compared with other accelerators in terms of power, speed, and area. Kuon et al. [50] have studied the gap between designs implemented in FPGA and ASIC. For 90 nm

B.S.C. Varma et al., *Architecture Exploration of FPGA Based Accelerators for BioInformatics Applications*, Springer Series in Advanced Microelectronics 55, DOI 10.1007/978-981-10-0591-6_2

technology, they report that the FPGA implementations are 5× slower and occupy 35× more area compared to ASIC implementations. They also report that dynamic power of FPGA implementations is 14 times that of ASIC. They also report that by using the HEBs in FPGAs, these ratios can be decreased. Even though ASICs provide high performance they are not popular as accelerators because ASIC implementations have long "design time" and do not offer much flexibility once the implementation is done. One more disadvantage of ASICs as accelerator is that the cost of producing them at low volumes is very high. As flexibility offered by ASICs is low, the demand for such specific accelerator chips is low but also their "life" is also low due to changes in standards as well as applications. FPGAs are cheaper compared to ASICs as they offer a lot of flexibility due to their reprogrammability. The hardware can be reconfigured for different kernel implementations. These advantages make FPGA one of the most popularly used accelerators.

Recently GPUs and CELL as accelerators are being extensively used as accelerators. These accelerators have a large number of floating point arithmetic units as cores which take advantage of single instruction multiple data (SIMD) processing. Many researchers have compared various accelerator architectures [16, 32, 35, 46]. Most of them report that GPUs and CELL are useful for floating point computations and FPGAs perform well on fixed point or integer computations. Further, FPGAs outperform other accelerators in some domains including cryptography.

2.1.1 Challenges in Designing Accelerators

The accelerators differ mainly on parameters such as price, power, productivity, performance, and ease of programming. Flexibility is an important aspect while deploying accelerators. FPGAs give more "configuration" flexibility vis-a-vis GPUs and CELL, but run at relatively lower clock speeds. The power consumption of GPUs is relatively high and can pose problem in cluster deployment.

Sanjay et al. [70] discuss the various issues related to the design of accelerators. They report that the main concern with accelerators is the lack of standard models in both architecture and software programming. This creates portability, scalability, and software management issues. A common programming language which allows ease of programming and standard interface design between host and accelerator to allow ease of data transfers across memory hierarchies can solve these problems.

The major obstacle for wider use of accelerators is that they require extra programming effort, which leads to long learning curves and hence result in increase in design time as well as cost. As the hardware of GPUs, CELL, and multicore CPUs are fixed and not designed from the perspective of bioinformatics applications, the software needs to be "tailored" and "partitioned" to utilize the resources efficiently for getting high performance. Each of the accelerators provides different programming interfaces and extensions. For designing an accelerator using FPGAs, we have to do a custom design of hardware.

2.2 FPGA-Based Acceleration

FPGAs are integrated circuits consisting of an array of logic blocks interconnected by routing resources which are programmable. Depending on their architecture they can be programmed either once or many times. The popular FPGA players presently in the market are Xilinx and Altera, which are based on SRAMs and thus easily reprogrammable [4, 92]. The other kind of FPGAs are based on anti-fuse, EPROM based [72]. Many other architectures based on emerging technologies have been proposed by many research groups [27, 83, 95]. The SRAM-based FPGAs are reprogrammable and hence are used as custom accelerators for various time-consuming applications. Knowing the underlying architecture of FPGAs, will help design better accelerators.

2.2.1 FPGA Architecture

A simple FPGA architecture is as shown in Fig. 2.1 [13]. The configurable logic blocks (CLBs) as well as input/output (I/O) blocks are programmable and all these are interconnected through programmable routing channels. A basic CLB is shown in Fig. 2.2. It consists of LUT, a flip flop and a mux. The mux is used to choose between direct output from the LUT or the registered LUT output. The respective output for an input combination is stored in the LUT. A combination of such CLB units is used to construct any desired logic circuit. A connect box is used to connect the CLB outputs to the routing channels. A connect box is used to interconnect various routing resources.

SRAM-based FPGAs are programmed using bitstreams. FPGA design flow is shown in Fig. 2.3. The flow shows the various steps involved in hardware implementation that culminates in generation of bitstream starting from HDL or schematics. High-level design languages like C are also being used to describe hardware. The

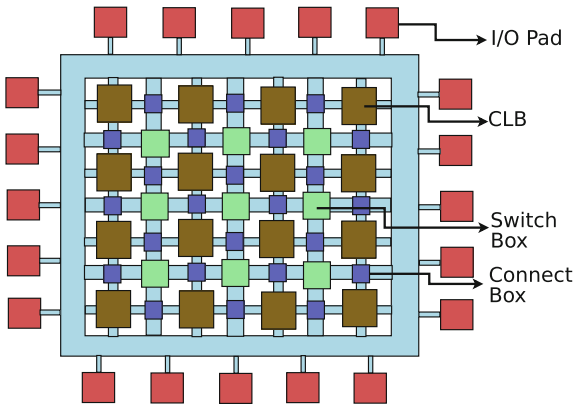

Fig. 2.1 FPGA architecture

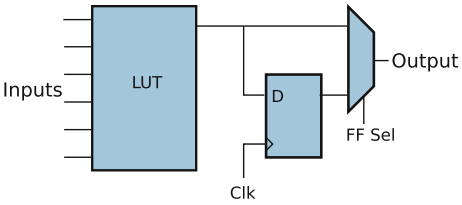

Fig. 2.2 Basic CLB structure

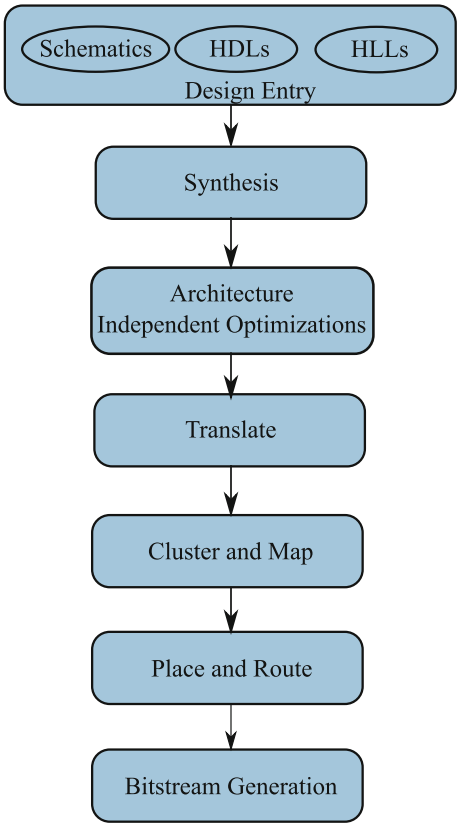

Fig. 2.3 FPGA design flow

synthesis tools do optimizations on the hardware described and convert it into gate level net-list. The translate tool packs these gates into LUTs. The place and route tool places the LUTs in an optimized way so as to have a less critical path. Typically, this is achieved by placing the LUTs representing the combinational logic (between two registers) close to each other. The routing tool defines the interconnectivity between the LUTs. This tool optimizes the use of long and short wires such that minimum number of switch blocks is used. Then a bitstream to program each of the LUTs and the interconnect are generated. The FPGA is programmed using this bitstream.

Lots of research have been carried out on FPGA architecture both by academia and industry. Several research groups have reported results on deciding the optimal LUT size [1, 34, 51, 75]. Larger LUTs reduce the interconnect delays but occupy more area and are slower because of the internal circuitry needed to decode the output. The optimal LUT size as reported by many researchers lies between 4 and 6. It is mostly dependent on the size of clustering of CLBs and the channel width. The present commercial FPGAs have LUT sizes up to 8. Xilinx has six input fracturable LUTs, whereas Altera has eight input fracturable LUTs [4, 92]. As the name suggests, "fracturable LUTs" can be broken and used as two smaller size LUTs.

Advances in FPGA-based reconfigurable computing (RC) technology over the past decade has allowed researchers to exploit performance, power, area, cost and versatility advantages in FPGA devices as compared to conventional microprocessor and ASICs [50]. More than a thousand-fold parallelism can be achieved for low-precision computation [37, 38]. Previously FPGAs were used for glue logic between incompatible systems, but now as technology has advanced much more logic can be implemented on an FPGA. Hence FPGAs are being used as standalone computing/logic units.

2.2.2 FPGA-Based Accelerators

Custom processors can be augmented to a base processor for accelerating computations. Xtensa processor from Cadence allows instruction set configurability [15]. Processors with custom instruction set can be developed using the tools. Custom accelerators are usually implemented on FPGAs as they are easily adaptable. Taking design decisions manually from a large number of choices based on complex trade-offs including reuse, area and gain is cumbersome and time consuming. Computer-aided design (CAD) tools have been developed that select optimal combination of accelerators by throughly searching the entire design space [71]. These tools reduce design time and make it easier for the designer to develop custom accelerators for different kind of applications hence reusing the same setup and reducing the cost. Recently many soft-core processors like Leon are available in public domain [33]. These are widely being used as multiprocessor systems on FPGAs. As the source code for these processors is available, the designer can modify the existing instruction set according to application requirements to improve performance.

Typically, FPGA cards or boards are plug-in cards that are plugged into slots available on the motherboard of computers. The boards contain one or more FPGAs. The FPGAs are connected to each other by direct wired connections. User constraints provided to the tools ensure that the hardware components in different FPGAs are connected to the right pins. The communication interfaces between host and FPGA are predefined during board design. Some boards use separate FPGAs for managing data through the interfaces. The current FPGA boards use PCIe interfaces and can be plugged into the PCIe slots available on CPUs [2, 42, 93]. Hardware implementation is done in HDLs. The bit files to program the FPGAs are generated using respective

FPGA tools. Typically FPGAs are programmed using these bit-files via application program interfaces (APIs) which wraps around a device driver. As device capacity of the FPGA changes, the API has to be modified to account for the granularity of functions that are implemented on them.

Even though FPGA-based accelerators give promising results, their use is not wide spread as they pose significant challenges due to limitations of application development tools and design methodologies [52]. The construction of accelerators consumes a huge amount of time and is very laborious. Tarek El-Ghazawi et al. did a research study focusing on investigating key challenges [28]. The authors tried to identify limitations of existing FPGA tools. In the report they discuss limitations that are based upon each of the four application development phases namely "Formulation," "Design," "Translation," and "Execution".

Critical design decisions are not made through algorithm design, architecture exploration and mapping but rather mostly on ad hoc basis. This causes costly iterations in the design flow. Many tools exist for designing accelerators from hardware description languages (HDLs) or HLLs, but need expertise in hardware for exploiting the FPGA resources properly. HLLs do not aid the designer to specify the concurrency required to fully exploit the features of FPGA-based systems. The lack of good co-design tools and languages is a concern and this leads to manual design partitioning. The manual design and partitioning is specific to the interfaces present in the accelerator card/board and hence it reduces portability and interoperability across platforms. OpenCL is one such language which attempts to solve this interoperability issue [82]. Many commercial vendors are supporting the use of this language [4, 67, 92]. Even though the EDA tools using this promises easy implementation, manual intervention is still required to get benefits for the specific accelerator [78].

Translation time is a major hurdle to FPGA synthesis. The place and route tools take a long time and sometimes are not very efficient and require designer interference to pack more logic. The execution challenges are that of run-time support for debugging. Most tools don't provide this feature. The tools for run-time reconfiguration are also not very efficient.

2.2.3 Hard Embedded Blocks in FPGAs

Typically, FPGA implementation occupies more silicon area and run at slower clock frequencies vis-a-vis ASICs. To implement large circuits using configurable logic in the FPGA may some times need more than one FPGA. Use of multiple FPGAs require more printed circuit board (PCB) space and more circuitry to manage the I/Os between the FPGAs. Synchronization is a matter of concern. To alleviate this problem, many of the commonly used sub-circuits in a design are available as HEBs in FPGAs. As these HEBs are custom designed and not implemented as CLBs in the FPGA, they occupy less silicon area and have higher operating frequency. Figure 2.4 shows a simplistic diagram of a modern FPGA consisting of DSP units as HEBs. As shown in the figure, additional HEBs are provided along with the CLBs and routing

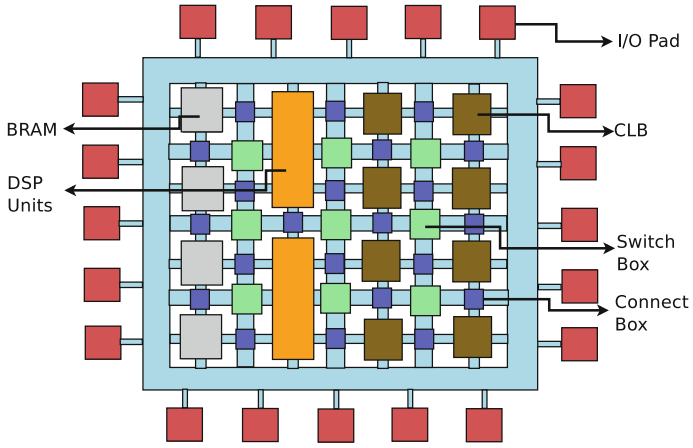

Fig. 2.4 Modern FPGA architecture

resources in the FPGA. Many a times, special routing resources are also provided for the HEBs. Modern FPGA contains memory controllers, PCIe controllers, FIFO controllers, serial transceivers, clock management, AES encryption units, ethernet MAC blocks, etc. as HEBs [92].

The HEBs in FPGAs can be broadly classified as fine-grained HEBs and the coarse-grained HEBs. The hardware blocks which are smaller in size and are used frequently are usually implemented as HEBs. A hardware design using a number of such fine-grained HEBs will be beneficial in terms of area, power and delay. These fine grained HEBs are typically distributed all over the area of the FPGA, and thus they can be closer to the blocks using them and hence reduce the interconnect delays. Some amount of configurability is provided to these blocks, so that a collection of them can be used to build larger hardware blocks. The coarser grained HEBs are hardware circuits which occupy significant amount of silicon area if they were implemented using the CLBs. Since these blocks are large, the configurability is less. As these blocks are pre-routed and do not use the interconnect blocks in the FPGA, they take less area and can run at high clock frequencies. Table 2.1 shows implementations of two hardware kernels on a Xilinx Virtex-6 FPGA. These were generated using Xilinx core-generator [90]. It can be observed that the equivalent silicon area used is significantly less compared to CLB only implementations. For example, the 64-bit signed multiplier unit uses 4265 slices, whereas the same implementation requires 16 DSP units whose equivalent area in terms of slices is 60, i.e., nearly two orders less.

Some of the fine-grained HEBs available in modern FPGAs are the block RAMs and carry-save arithmetic blocks. Almost all the applications require memory to store the intermediate results. These intermediate results are either stored in registers or in memory blocks. The advantage of storing data in the registers is that they can be written and read by many compute blocks, but they occupy more silicon area.

Table 2.1 Resources occupied by cores using only slices and using DSP units [92]

Core	Slices	DSP	Eq. slices
64-bit signed multiplier	4256	0	4256
	0	16	80
48-bit cordic unit	9138	0	9138
	0	12	60

Bit-level operations can be easily done on registers. A disadvantage of using registers is that in FPGA implementations consisting of a lot of wide bit-width registers, a lot of routing resources will be needed and thus may cause the operating frequency to reduce. The memory blocks are preferred when the data from the block is needed by very few compute units. Multi-ported memories are typically used in such scenarios, but they occupy more area and hardware support to keep a check on the "read after write" (RAW) and "write after read" (WAR) hazards. Modern FPGAs have memory blocks as HEBs. Embedded block RAMs were proposed and studied by Wilton et al. [66, 86–88]. The authors propose various ways to build memories as HEBs. Memories of various sizes have to be built based on the application. Choosing the right size of memory block is an important issue in FPGAs. Larger blocks will cause the memory to be unused, whereas the smaller blocks will cause routing congestion. Due to unavailability of multiple ports, larger blocks may sometimes cause the data-starvation in the compute units. The authors propose design of smaller memory blocks so that they can be combined to form larger blocks of memory. They also show that hardware implementations can benefit from logic circuits implemented using these memories as LUTs. Xilinx FPGAs provide 18 and 32 kb embedded block RAMs running at 550 MHz [89]. Altera also provides embedded memories named as MRAMs [5]. They are configurable and can be used to build larger memory blocks.

Various enhancements to the FPGA architecture have been done to improve the hardware implementation of arithmetic operations. The carry chains are provided along with the CLBs as HEBs to build efficient adders. The adder implementations typically require 2 LUTs to implement logic for carry and the sum. The use of carry chain reduces this as well as the interconnect delays. Parandeh-Afshar et al. [69] propose a way to improve the performance of carry-save arithmetic in FPGAs using generalized parallel counters. They show 1.8× improvement in energy consumption. Many of the FPGAs provide digital signal processing (DSP) units as HEBs [3, 91]. These DSPs units are configurable and can be used for performing addition, multiplication as well as multiply accumulator operations. The newer DSP units also support other bit-level operations. Beauchamp et al. [9] have proposed floating point units as HEBs. The authors implement both single-precision and double-precision floating point units as HEBs in FPGAs and show 55 % area benefits and 40.7 % increase in clock frequency. Flexible embedded floating point unit, which can be used both as single-precision and double-precision floating point units, has been proposed by Chong et al. [21].

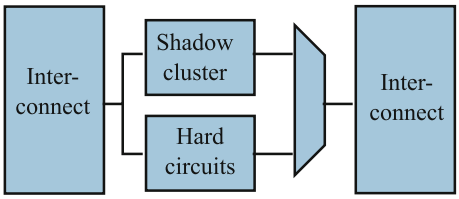

Fig. 2.5 Shadow clusters for performance benefits [44]

Configurable hard blocks have advantages as they can be used in wider applications without wasting the silicon area. The fabrics which have HEBs are expensive since the user has to pay for the cores available for HEBs even if they do not use them. This also effectively reduces the reconfigurable area for the same silicon area, if the HEBs are not used. Jamieson et al. [44] propose shadow circuits to improve the performance of designs without losing the flexibility of the FPGAs. The basic idea is shown in Fig. 2.5. Here the shadow clusters and the hard circuits are separated using the mux. The whole block is incorporated as HEB in FPGA fabric. The key idea is to use the silicon area efficiently when the hard circuits can not be directly used. The shadow circuits enable partial use of HEBs and thus increases the overall fabric area usage. They also implement CAD tools to support this type of mapping.

A proper methodology is needed to evaluate the HEBs to be incorporated in the FPGA fabric, in order to use the silicon area efficiently. Tool-flows to evaluate HEBs are discussed in more detail in Chap. 3.

One of the key areas which could benefit by using FPGA-based acceleration is bioinformatics. Applications in bioinformatics domain are compute intensive. Data-level parallelism can be exploited by hardware accelerators to speedup software implementations. In the next section we review some important concepts in the bioinformatics domain and introduce the need and scope of accelerators for two key algorithms.

2.3 Bioinformatics

Bioinformatics is an interdisciplinary subject consisting partly of molecular biology and partly of computers. It deals with use of computers to analyze and organize biological data. This field has advanced so rapidly that nowadays chemists, mathematicians, statisticians, and computer engineers work as a team to solve the problems [24]. DNA sequencing costs have been reduced by increasing throughput by the use of massively parallel DNA sequencing systems. These systems are capable of determining the sequence of huge numbers of different DNA strands at one time. These "next generation sequencing" (NGS) systems allow millions of reads to be gathered in a single experiment [40]. These sequencing techniques have led to exponential increase in data. NGS aspires to find solutions in genetic analysis. Understanding and analyzing this large volume of data is of immense importance to biologists. This

large volume creates challenges in acquisition, management and analysis of data. The major research areas in bioinformatics are sequence alignment, gene structure prediction, construction of phylogenetic trees, protein folding and drug design [23]. Many software tools have been developed to aid biologists in their respective fields [6, 31, 49, 59, 74]. Computer scientists and biologists are working together to overcome the challenges of rapid interpretation of the data. We describe two important bioinformatics applications that we address in this book, namely protein docking and genome assembly.

2.3.1 Protein Docking

Study of inter-molecular interactions has various applications in biology, especially in the process of drug design. An important and time-consuming part of drug design process is the "virtual screening", which is a computational technique used to find the drug molecules which binds well with a particular protein molecule from a large library of probable drug molecules [47, 60]. "Protein docking" is a software application which is used to do this screening process. The docking application scores the drug molecules based on the binding parameters. The docking program ranks the protein–ligand complex based on scoring functions. Scoring functions predict the strength of the complex. The highly ranked drugs or drug conformations are then checked to see if they are suitable for human beings. Even though the computer-based virtual screening is faster than chemical processes, most of the docking tools perform computationally intensive calculations and take large amount of time to execute on CPUs. The medical field is gradually moving towards personalized medicines and hence there will be a need to do a virtual screening on a large scale. Jenwitheesuk et al. [45] have proposed that computational screening of small drug-like molecules against multiple proteins will increase the efficiency for finding drugs for drug-resistant diseases. Implementing such a design for drug design will take significant amount of time and accelerating docking program becomes very important for making such a system pragmatic.

An example of protein complex is shown in Fig. 2.6. The diagram shows a ligand which is bound to the protein. Here the protein–ligand complex is in a state where overall free energy of the system is minimum. Docking will help find the orientation of the ligand so as to give such minimized free energy state system.
There are three types of docking [36]:

1. Rigid molecular docking
2. Flexible docking
3. Combination of both rigid and flexible docking

Rigid docking is a type of modeling in which the internal geometry of the interacting elements are kept fixed and relative orientations are varied. Changes in the internal geometry might occur during the formation of complex and docking which takes this into consideration is called flexible docking. The combination of these two is also

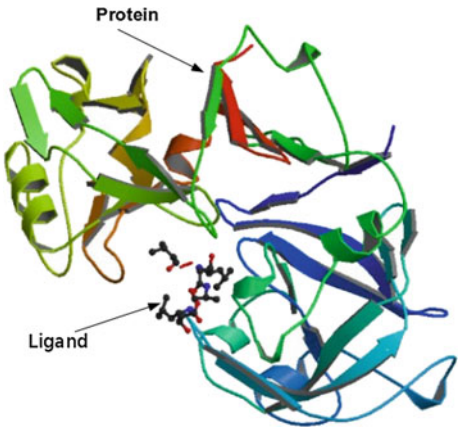

Fig. 2.6 Biological assembly image for docked complex [30]

used, where a small subset of possible conformational changes are taken into account to reduce the computation time. Some of the tools used for docking are FTDock [81], Autodock [62], PIPER [49] and Hex [26], ZDock [17], F2Dock [7].
Typical docking application involves the following steps:

1. Discretize the molecules
2. Rotating the molecules
3. Shape complementarity score calculations
4. Electrostatic complementarity score calculations

The discretization process involves placing the molecule in a 3D grid and noting the position of each atom in that molecule. This process is done for both the proteins in case of protein–protein docking, and both ligand and protein in case of protein–ligand docking. The total grid size is dependent on the size of molecule. The grid should cover the whole of the larger molecule. Resolution of the grid decides the accuracy of the experiment. Smaller grid resolution gives more accurate results but require more computations vis-a-vis large grid resolution. The discretized molecule is stored as a 3D matrix.

To study the different orientations of the molecules and to study their binding, systematic rotation of the protein or ligand molecules is carried out. Typically, larger molecule's position/orientation in the grid is kept constant and the smaller molecule is rotated around the axes. The various scores are calculated for each of the rotated smaller molecule. Here also, the resolution of rotation is chosen based on the accuracy required. For example for a resolution of 6°, there are 54,000 rotations possible where for 12° rotation 13,500 rotations are possible. Some of the approaches calculate other scores based on bonding.

Shape complementarity score and electrostatic complementarity score are calculated using correlation functions. For electrostatic complementarity the Coulombic forces on each of the atom by other atom is stored in the 3D matrix. These correlation functions are mostly convolution of the two matrices representing the

discretized matrices. The convolution function 'C' of two matrices 'A' and 'B' is given by (2.1).

$$C(i, j, k) = \sum_{l=1}^{L} \sum_{m=1}^{M} \sum_{n=1}^{N} A(l, m, n) * B(i - l, j - m, k - n) \tag{2.1}$$

The calculation of the convolution function on general purpose processors is expensive as it involves lots of multiplications. In many of the docking applications, FFTs are used to reduce the number of multiplications [7, 49].

2.3.1.1 Need for Speedup and Use of Accelerators

Even though a significant number of multiplications are reduced by use of FFT to $\log_2(N)$ vis-a-vis direct convolution which has N^3 multiplication, 3D-FFT still takes a very long time to execute on processors. The run-times for the various docking applications are shown in Table 2.2 which are reported by Dave et al. in [73]. The authors compare the run-time of Hex docking application [26] which is their own implementation, with the run-times of two other docking applications—ZDock [17] and PIPER [49]. The docking was carried out for Kallirein A/BPT1 complex. The protein molecules grid size was 128 and the ligand molecules grid size was 32. The processor used was Intel Xeon processor and the GPU used was Nvidia GTX-285. The ZDock application makes use of convolution function directly. ZDock does some optimizations for carrying out convolution function on sparse matrices. Piper [49] uses 3D-FFT for accelerating convolution function, whereas Hex docking application uses 3D-FFT in spherical coordinates to accelerate computations. The run-times of using 1D-FFT is also shown. It can be seen that there is significant reduction in execution time when GPUs are used. For example Hex docking computing 3D-FFTs gives speed-ups of about 2.7×. The GPUs carry out the FFT computation in parallel on the multiple floating point units and hence exploit the parallelism available in the application. Bharat et al. have attempted to accelerate PIPER docking application using FPGAs [10]. They report that for smaller sized molecules (grid size <16) FPGAs perform better than GPUs. They also report that GPUs perform better for

Table 2.2 Run-times of various docking programs [73]

Docking	Docking	Run-time (s)	
Software	Type	CPU	GPU
ZDock	3D	7172	–
Piper	3D	468,625	26,372
Hex	1D	676	15
	3D	224	84

docking of larger molecules. They report that FPGA implementation based on direct correlation function is better in FPGAs and FFT-based implementations are better suited for GPUs.

2.3.2 Genome Assembly

The basic building block of all organisms is the *cell*. All the cells (irrespective of the size of organism) have a nucleus which carries a genetic material known as *deoxyribonucleic acid* (DNA). DNA holds the hereditary information and is responsible for the controlling and functioning of the organism. DNA is made up of four bases: adenine (A), guanine (G), cytosine (C), and thymine (T). Adenosine (A) pairs with thymine (T), and cytosine (C) pairs with guanine (G) forming *base pairs* (bp). This pairing is due to weak hydrogen bonding and is the basis for DNA replication. A segment of a DNA molecule can be written using the first letter of the bases it contains (eg., ...TACGTAG...).

The complete set of all genes along with noncoding deoxyribonucleic acid (DNA) in an organism is called a *genome*. The study of genomes of various organisms is known as genomics. It has a lot of applications in medicine, biotechnology, anthropology, forensics and synthetic biology. Also, comparative study of different genomes is helpful for evolutionary studies. Increasingly, genomics is also being used to study the contribution of genes in many diseases and is aiding in the development of personalized drugs. Hence, genome construction is very important, which helps considerably in the study of various biological processes in an organism.

DNA sequencing technology helps in generating the data needed for construction of genomes. Recently, next-generation sequencing (NGS) platforms are being used for DNA sequencing. These platforms generate short fragments called "*reads*" of length ranging from thirty-five to few hundreds of base pairs. These reads are part of a large genome containing millions of base pairs (the size of the human genome $= 3 \times 10^9$ bp). The NGS platforms generate large amounts of data at very low cost and at a greater speed when compared to older platforms [65].

The large amount of data has posed many challenges for computer scientists who develop softwares to analyze these data. New algorithms and data structures have been proposed to speed up the analysis [8, 58]. Databases have been created and statistical analysis programs have been developed for retrieving specific information from this data. *Sequence assembly* is a computational biology problem where the reads generated from the NGS machine are used to build the whole genome.

An example of assembly is shown in Fig. 2.7. A biological sample is preprocessed and given to a sequencing machine, which generates a set of short reads. These short reads are assembled to construct the genome. The 'T' in the read CTGTGTGT*T*, is an error as the exact match to the genome at that position was supposed to be 'C'. The error could be identified as the frequency of occurrence of 'C' at that position is more than that of 'T'. The error can occur during the sequencing process by the sequencing machine. Error can also occur while assembling the genome from the

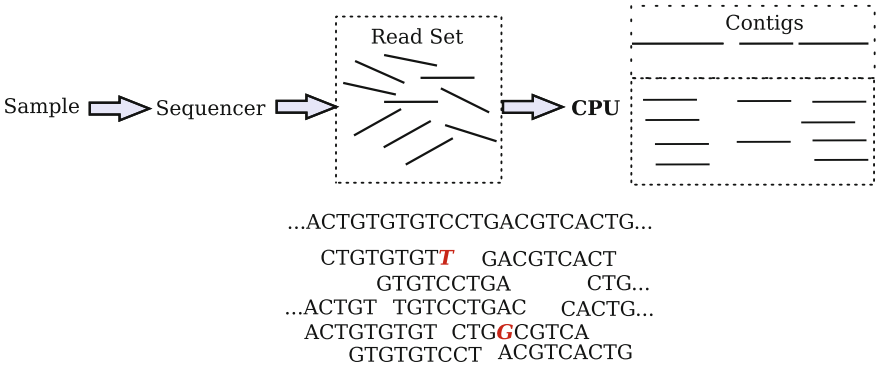

Fig. 2.7 NGS assembly: the DNA sample is given to sequencer, which generates read-set. The read-set is processed in CPUS to generate contigs

short reads, where the read is falsely mapped to a particular location of the genome. Due to these errors, genome assembly problem is more difficult to solve than the well studied shortest super-string problem [65].

2.3.2.1 Genome Assembly Softwares

Many types of software have been developed to do assembly [61]. Algorithms are modified in order to alleviate some of the complexities involved in the assembly and to execute efficiently on processors. Assembly software solutions can be divided into two categories; mapping based comparative assembly and de novo assembly. In the former method, assembly is done by mapping the reads to an already pre-existing reference genome. Even though the genomes of a particular organism contain lots of similarities, there are certain dissimilar regions which makes each organism unique. These dissimilar regions are of interest to biologists as they show particular behavior unique to that individual organism. Mapping the reads to a pre-existing reference genome might cause this uniqueness to be destroyed and hence the software assemblers allow certain amount of mismatches and gaps. Some of the mapping based assembly programs are SOAP [55], MAQ [54], Bowtie [53], and RMAP [79].

The later method is called de novo assembly where the information is extracted from the reads and their overlaps. De novo assembly is a type of assembly process where genome is constructed without using a reference genome. It is the only way to construct a genome if the reference genome does not exist. As the shorter reads have less overlap information, the reads are generated with much more coverage in order to construct the genome. The overlap information from the reads is used to construct the contiguous consensus sequences known as *contigs*. The genome is constructed using these contigs using a process known as scaffolding. Some of the de novo assembly software programs are Velvet [94], Edena [39], PerM [18], BFAST [41] and Minia [20]. De novo assembly takes more computational time than

Table 2.3 Run-time of various de novo genome assembly softwares [12]

Software application	Number of cores	RAM	Run-time
Hyda [63]	48	512 GB	14 h
Abyss [77]	12	4 GB each	13 h
Allpaths-LG [14]	48	512 GB	151–215 h
Velvet [94]	32	300–500 GB	3.5 weeks
Celera [64]	32	256 GB	9.5 days
SOAP de novo [57]	24–32	110–150 GB	48–72 h
Newbler [76]	12	130 GB	18 h
Ray [11]	256	768 GB	36–72 h
Monument [19]	16	140 GB	1 day

mapping based assembly. Since the mapping based assembly includes a pre-existing reference genome during mapping, the assembly process is biased and hence in certain situations bioinformaticians prefer to use de novo based assembly.

Clearly, like most of the bioinformatics applications, NGS assembly is also data dominated. Even though compute infrastructure ranging from server racks to cloud farms exist for solving these problems, the time taken is enormous. For example assembly of human genome using PASHA software took around 21 h on a 8-core workstation with 72 GB memory [56]. The run-times of various assembler programs as reported by Bradnam et al. [12] are shown in Table 2.3. The table shows the number of cores used, maximum RAM usage in most of the cases (authors report RAM available in the system in some cases) and the run-time. The run-times were reported for assembling snake, fish and bird genomes. They attempt to construct the genome for the first time and since they do not have any reference genome they use de novo genome assembly. It can be seen that the run-time taken by different softwares vary from 10 h to 3.5 weeks. Each of the assembler softwares have their own pipeline/flow to construct the genome, starting from processing of the biological sample. The selection of assembler application is also based on the sequencing machine used for generating the reads. For example, 454-sequencing machine provide a tool called Newbler which is a proprietary application that works best for the data it produces [76]. As most of the applications take significant amount of time to execute, there is a need to accelerate them.

2.3.2.2 FPGA-Based Acceleration

Fernandez et al. [79] have reported FPGA acceleration of NGS mapping [29]. They show speedups up to 4× over RMAP software. The reads are stored in registers in the FPGA and the reference sequence is streamed through shift-registers for comparison. Multiple reads are compared with the reference genome sequence and hence parallelism is exploited to get speed-ups over software application. Knodel et al. [48]

have also accelerated NGS short read mapping. They use a similar approach used by Fernandez et al. [29] and show that the performance is comparable to many software applications like Bowtie [53]. Tang et al. [84] report 42× speedup over software PerM [18] using FPGAs. PerM uses comparative assembly. They construct a pipeline of Processing Elements (PEs) in FPGAs. The reference sequence is compared with the reads in the PEs. The number of mismatches is counted and if it is less than a threshold, the position of the read is reported. Pairwise sequence comparison is executed in parallel and thus speedups are obtained software implementations. Olson et al. [68] has also shown acceleration of short read mapping on FPGA. The authors show 250× improvement over BFAST software [41] and 31× when compared to Bowtie [53]. All short sequences of fixed length (eg., 22) known as "seeds" in the reference sequence are indexed in a table. Each of the "seeds" present in the "reads" are searched in the index table. If the seed is found in table, Smith–Waterman algorithm is used to compare the read with the indexed portion of the reference sequence [80]. The hashing and the Smith–Waterman algorithm is implemented in FPGA. Speed-ups over software is obtained because of the hash function, which reduces the comparison and due to the fast implementation of dynamic programming (Smith–Waterman algorithm) on FPGAs. The Convey Computer firm has developed the Convey GraphConstructor (CGC), which use FPGAs to accelerate de novo assembly [25]. They show speedups of 2.2× to 8.4×. There is very little work reported in literature on FPGA acceleration of de novo genome assembly. It can be seen that there is a need to accelerate many assembly applications.

2.4 Summary

In this chapter we have discussed the various accelerator architectures. We have discussed FPGA-based accelerators in detail. We have described the basic architecture of FPGA and how they have evolved to include HEBs. We report some of the research works related to HEBs. We have introduced some basic concepts of bioinformatics and describe protein docking and genome assembly which are two important bioinformatics applications. We have discussed works related to accelerating these applications specifically using FPGAs. With this background, the DSE is explained in the next few chapters.

References

1. Ahmed, E., Rose, J.: The effect of LUT and cluster size on deep-submicron FPGA performance and density. IEEE Trans. Very Large Scale Integr. VLSI Syst. **12**(3), 288–298 (2004)
2. Alpha-Data: Alpha-Data FPGA Boards. http://www.alpha-data.com/ (2015)
3. ALTERA: Altera dsps. http://www.altera.com/technology/dsp/dsp-index.jsp (2015)
4. Altera: Altera FPGAs. http://www.altera.com (2015)

5. ALTERA: Altera mrams. http://www.altera.com/technology/memory/embedded/mem-embedded.html (2015)
6. Altschul, S.F., Madden, T.L., Schffer, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J.: Gapped blast and psiblast: a new generation of protein database search programs. Nucleic Acids Res. **25**(17), 3389–3402 (1997)
7. Bajaj, C., Chowdhury, R., Siddavanahalli, V.: F2Dock: fast Fourier protein-protein docking. IEEE/ACM Trans. Comput. Biol. Bioinform. **8**(1), 45–58 (2011)
8. Baker, M.: Next generation sequencing: adjusting to data overload. Nat. Methods 495–499 (2010)
9. Beauchamp, M.J., Hauck, S., Underwood, K.D., Hemmert, K.S.: Embedded floating-point units in FPGAs. In: ACM/SIGDA International Symposium on FPGAs (2006)
10. Bharat, S., Herbordt, M.C.: GPU acceleration of a production molecular docking code. In: GPGPU (2009)
11. Boisvert, S., Laviolette, F., Corbeil, J.: Simultaneous assembly of reads from a mix of high-throughput sequencing technologies. J. Comput. Biol. **17**(11), 1519–1533 (2010)
12. Bradnam, K.R., Fass, J.N., Alexandrov, A., Baranay, P., et al.: Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. GigaSci. **2**(1) (2013)
13. Brown, S., Rose, J.: Architecture of FPGAs and CPLDs: a tutorial. IEEE Des. Test Comput. **13**, 42–57 (1996)
14. Butler, J., MacCallum, I., Kleber, M., Shlyakhter, I.A., Belmonte, M.K., Lander, E.S., Nusbaum, C., Jaffe, D.B.: ALLPATHS: de novo assembly of whole-genome shotgun microreads. Genome Res. **18**(5), 810–820 (2008)
15. CADENCE: Tensilica customizable processors. http://ip.cadence.com/ipportfolio/tensilica-ip (2015)
16. Che, S., Li, J., Sheaffer, J., Skadron, K., Lach, J.: Accelerating compute-intensive applications with gpus and fpgas. In: Symposium on Application Specific Processors, 2008. SASP 2008 (2008)
17. Chen, R., Li, L., Weng, Z.: ZDOCK: an initial-stage protein-docking algorithm. Proteins **52**(1), 80–87 (2003)
18. Chen, Y., Souaiaia, T., Chen, T.: PerM: efficient mapping of short sequencing reads with periodic full sensitive spaced seeds. Bioinformatics **25**(19), 2514–2521 (2009)
19. Chikhi, R.: Monument assembler. http://www.irisa.fr/symbiose/people/rchikhi/monument.html (2015)
20. Chikhi, R., Rizk, G.: Space-Efficient and Exact de Bruijn Graph Representation Based on a Bloom Filter. In: Raphael, B., Tang, J. (eds.) Algorithms in Bioinformatics. Lecture Notes in Computer Science, vol. 7534, pp. 236–248. Springer, Berlin Heidelberg (2012)
21. Chong, Y.J., Parameswaran, S.: Flexible multi-mode embedded floating-point unit for field programmable gate arrays. In: ACM/SIGDA International Symposium on FPGAs (2009)
22. Chung, E., Milder, P., Hoe, J., Mai, K.: Single-chip heterogeneous computing: does the future include custom logic, FPGAs, and GPGPUs? In: Annual IEEE/ACM International Symposium on Microarchitecture (MICRO) (2010)
23. Cohen, J.: Bioinformatics an introduction for computer scientists. ACM Comput. Surv. **36**, 122–158 (2004)
24. Cohen, J.: Computer science and bioinformatics. Commun. ACM **48**, 72–78 (2005)
25. Convey Computer: Convey GraphConstructor. http://www.conveycomputer.com (2015)
26. David, Ritchie, W., Ritchie, D.: Hex 6.0 user manual protein docking using spherical polar fourier correlations
27. Devadoss, R., Paul, K., Balakrishnan, M.: p-qca: a tiled programmable fabric architecture using molecular quantum-dot cellular automata. J. Emerg. Technol. Comput. Syst. **7**(3), 13:1–13:20 (2011)
28. El-Ghazawi, T., George, A.D., Gonzalez, I., Lam, H., Merchant, S., Saha, P., Smith, M., Stitt, G., Alam, N., El-Araby, E., Holland, B., Reardon, C.: Exploration of a Research Roadmap for Application Development and Execution on Field-Programmable Gate Array (FPGA)-Based Systems. Technical Report, Defense Technical Information Center (2008)

29. Fernandez, E., Najjar, W., Harris, E., Lonardi, S.: Exploration of short reads genome mapping in hardware. In: International Conference on FPL, pp. 360–363 (2010)
30. Fujinaga, M., Chernaia, M.M., Tarasova, N.I., Mosimann, S.C., James, M.N.: Crystal structure of human pepsin and its complex with pepstatin. Protein Sci. **4** (1995)
31. Gabb, H.A., Jackson, R.M., Sternberg, M.J.E.: Modelling protein docking using shape complementarity, electrostatics and biochemical information. J. Mol. Biol. **272** (1997)
32. Gac, N., Mancini, S., Desvignes, M., Houzet, D.: High speed 3D tomography on CPU, GPU, and FPGA. EURASIP J. Embed. Syst. **2008**, 5:1–5:12 (2008)
33. Gaisler: LEON processors. http://www.gaisler.com/index.php/products/processors/ (2015)
34. Gao, H., Yang, Y., Ma, X., Dong, G.: Analysis of the effect of LUT size on FPGA area and delay using theoretical derivations. In: Sixth International Symposium on Quality of Electronic Design, 2005. ISQED 2005, pp. 370–374 (2005)
35. Grozea, C., Bankovic, Z., Laskov, P.: Facing the multicore-challenge. chap. FPGA vs. Multicore CPUs vs. GPUs: Hands-on Experience with a Sorting Application, pp. 105–117. Springer-Verlag, Berlin, Heidelberg (2010)
36. Halperin, I., Ma, B., Wolfson, H., Nussinov, R.: Principles of docking: an overview of search algorithms and a guide to scoring functions. Proteins Struct., Funct., Bioinf. **47**(4), 409–443 (2002)
37. Hauck, S., DeHon, A.: Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation. Morgan Kaufmann, Systems on Silicon (2007)
38. Herbordt, M.C., VanCourt, T., Gu, Y., Sukhwani, B., Conti, A., Model, J., DiSabello, D.: Achieving high performance with FPGA-based computing. Computer **40**(3), 50–57 (2007)
39. Hernandez, D., François, P., Farinelli, L., Østerås, M., Schrenzel, J.: De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer. Genome Res. **18** (2008)
40. Hert, D.G., Fredlake, C.P., Barron, A.E.: Advantages and limitations of next-generation sequencing technologies: a comparison of electrophoresis and non-electrophoresis methods. Electrophoresis **29**(23), 4618–4626 (2008)
41. Homer, N., Merriman, B., Nelson, S.F.: BFAST: an alignment tool for large scale genome resequencing. PLoS ONE **4** (2009)
42. Inc., D.: Digilent FPGA Boards. https://www.digilentinc.com/ (2015)
43. ITRS: The International Technology Roadmap for Semiconductors. http://www.itrs.net (2015)
44. Jamieson, P., Rose, J.: Enhancing the area efficiency of fpgas with hard circuits using shadow clusters. IEEE Trans. VLSI Syst. **18**(12), 1696–1709 (2010)
45. Jenwitheesuk, E., Horst, J.A., Rivas, K.L., Voorhis, W.C.V., Samudrala, R.: Novel paradigms for drug discovery: computational multitarget screening. Trends Pharmacol. Sci. **29**(2), 62–71 (2008)
46. Kapre, N., DeHon, A.: Performance comparison of single-precision SPICE model-evaluation on FPGA, GPU, cell, and multi-core processors. In: International Conference on Field Programmable Logic and Applications, 2009. FPL 2009, pp. 65–72 (2009)
47. Kitchen, D.B., Decornez, H., Furr, J.R., Bajorath, J.: Docking and scoring in virtual screening for drug discovery: methods and applications. Nat. Rev. Drug Discov. **3**(11), 935–949 (2004)
48. Knodel, O., Preusser, T., Spallek, R.: Next-generation massively parallel short-read mapping on FPGAs. In: IEEE International Conference on ASAP, pp. 195–201 (2011)
49. Kozakov, D., Brenke, R., Comeau, S.R., Vajda, S.: PIPER: an FFT-based protein docking program with pairwise potentials. Proteins (2006)
50. Kuon, I., Rose, J.: Measuring the gap between FPGAs and ASICs. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **26**(2), 203–215 (2007)
51. Kuon, I., Rose, J.: Area and delay trade-offs in the circuit and architecture design of FPGAs. In: Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays. FPGA '08, pp. 149–158. ACM, New York, NY, USA (2008)
52. Kuon, I., Tessier, R., Rose, J.: FPGA architecture: survey and challenges. Found. Trends Electron. Des. Autom. **2**, 135–253 (2008)

53. Langmead, B., Trapnell, C., Pop, M., Salzberg, S.: Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biol. **10** (2009)
54. Li, H., Ruan, J., Durbin, R.: Mapping short DNA sequencing reads and calling variants using mapping quality scores. Genome Res. **18**(11), 1851–1858 (2008)
55. Li, R., Li, Y., Kristiansen, K., Wang, J.: SOAP: short oligonucleotide alignment program. Bioinformatics **24**(5), 713–714 (2008)
56. Liu, Y., Schmidt, B., Maskell, D.: Parallelized short read assembly of large genomes using De Bruijn graphs. BMC Bioinf. **12**(1), 354–363 (2011)
57. Luo, R., Liu, B., Xie, Y., Li, Z., Huang, W., Yuan, J., He, G., Chen, Y., Pan, Q., Liu, Y., Tang, J., Wu, G., Zhang, H., Shi, Y., Liu, Y., Yu, C., Wang, B., Lu, Y., Han, C., Cheung, D.W., Yiu, S.M., Peng, S., Xiaoqian, Z., Liu, G., Liao, X., Li, Y., Yang, H., Wang, J., Lam, T.W., Wang, J.: SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. Gigascience **1**(1), 18 (2012)
58. Mardis, E.R.: Next generation DNA sequencing methods. Ann. Rev. Genomics Hum. Genet. **9**, 387–402 (2008)
59. McCollum, J.M., Peterson, G.D., Cox, C.D., Simpson, M.L.: Accelerating gene regulatory network modeling using grid-based simulation. Simulation **80**(4–5), 231–241 (2004)
60. McInnes, C.: Virtual screening strategies in drug discovery. Curr. Opin. Chem. Biol. **11**(5), 494–502 (2007)
61. Miller, J.R., Koren, S., Sutton, G.: Assembly algorithms for next-generation sequencing data. Genomics **95**(6), 315–327 (2010)
62. Morris, G.M., Huey, R., Lindstrom, W., Sanner, M.F., Belew, R.K., Goodsell, D.S., Olson, A.J.: J. Comput. Chem
63. Movahedi, N., Forouzmand, E., Chitsaz, H.: De novo co-assembly of bacterial genomes from multiple single cells. In: 2012 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pp. 1–5 (2012)
64. Myers, E.W., Sutton, G.G., Delcher, A.L., Dew, I.M., Fasulo, D.P., et al.: A whole-genome assembly of Drosophila. Science **287**(5461), 2196–2204 (2000)
65. Nagarajan, N., Pop, M.: Sequence assembly demystified. In: Nature Reviews Genetics, pp. 157–167 (2013)
66. Ngai, T., Rose, J., Wilton, S.J.E.: An sram-programmable field-configurable memory. In: Proceedings of the IEEE Custom Integrated Circuits Conference, 1995, pp. 499–502 (1995)
67. NVIDIA: Nvidia GPGPU. http://www.nvidia.com (2015)
68. Olson, C., Kim, M., Clauson, C., Kogon, B., Ebeling, C., Hauck, S., Ruzzo, W.: Hardware acceleration of short read mapping. In: IEEE Symposium on FCCM, pp. 161–168 (2012)
69. Parandeh-Afshar, H., Verma, A., Brisk, P., Ienne, P.: Improving fpga performance for carry-save arithmetic. IEEE Trans. Very Large Scale Integ. VLSI Syst. **18**(4), 578–590 (2010)
70. Patel, S., Hwu, W.W.: Guest editors' introduction: accelerator architectures. IEEE Micro **28**(4), 4–12 (2008)
71. Pothineni, N., Kumar, A., Paul, K.: Exhaustive enumeration of legal custom instructions for extensible processors. In: VLSID '08: Proceedings of the 21st International Conference on VLSI Design, pp. 261–266. IEEE Computer Society, Washington, DC, USA (2008)
72. QuickLogic: QuickLogic FPGAs. http://www.quicklogic.com/ (2015)
73. Ritchie, D.W., Venkatraman, V.: Ultra-fast FFT protein docking on graphics processors. Bioinformatics **26**(19), 2398–2405 (2010)
74. Rizk, G., Lavenier, D.: Gassst: global alignment short sequence search tool. Bioinformatics **26**(20), 2534–2540 (2010)
75. Rose, J., Gamal, A.E., Member, S., Sangiovanni-vincentelli, A.: Architecture of field-programmable gate arrays: the effect of logic block functionality on area efficiency. Proc. IEEE **25**, 1217–1225 (1990)
76. Sequencing, R.: 454 Sequencing. http://www.454.com/products/analysis-software/ (2015)
77. Simpson, J., Wong, K., Jackman, S., Schein, J., Jones, S., Birol, I.: ABySS: a parallel assembler for short read sequence data. Genome Res. **19**, 1117 (2009)

78. Singh, D.P., Czajkowski, T.S., Ling, A.C.: Harnessing the power of fpgas using altera's opencl compiler. In: Hutchings, B.L., Betz, V. (eds.) FPGA, pp. 5–6. ACM (2013)
79. Smith, A.D., Xuan, Z., Zhang, M.Q.: Using quality scores and longer reads improves accuracy of solexa read mapping. BMC Bioinf. **9**, 128 (2008)
80. Smith, T., Waterman, M.S.: Identification of common molecular subsequences. J. Mol. Biol. **147**(3), 195–197 (1981)
81. Sternberg, M.J.E., Aloy, P., Gabb, H.A., Jackson, R.M., Moont, G., Querol, E., Aviles, F.X.: A computational system for modeling flexible protein-protein and protein-DNA docking. In: Proceedings of the 6th International Conference on Intelligent Systems for Molecular Biology, pp. 183–192 (1998)
82. Stone, J., Gohara, D., Shi, G.: Opencl: a parallel programming standard for heterogeneous computing systems. Comput. Sci. Eng. **12**(3), 66–73 (2010)
83. Tabula: Tabula FPGAs. http://www.tabula.com/ (2015)
84. Tang, W., Wang, W., Duan, B., Zhang, C., Tan, G., Zhang, P., Sun, N.: Accelerating millions of short reads mapping on a heterogeneous architecture with fpga accelerator. Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 0, pp. 184–187 (2012)
85. Venkatesh, G., Sampson, J., Goulding, N., Garcia, S., Bryksin, V., Lugo-Martinez, J., Swanson, S., Taylor, M.B.: Conservation cores: reducing the energy of mature computations. SIGARCH Comput. Archit. News **38**(1), 205–218 (2010)
86. Wilton, S.J.E.: Embedded memory in fpgas: recent research results. In: 1999 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, pp. 292–296 (1999)
87. Wilton, S.J.E., Rose, J., Vranesic, Z.: The memory/logic interface in fpgas with large embedded memory arrays. IEEE Trans. Very Large Scale Integr. VLSI Syst. **7**(1), 80–91 (1999)
88. Wilton, S.J.E., Rose, J., Vranesic, Z.G.: Architecture of centralized field-configurable memory. In: Proceedings of the 1995 ACM Third International Symposium on Field-programmable Gate Arrays. FPGA '95, pp. 97–103. ACM, New York, NY, USA (1995)
89. XILINX: Xilinx brams (2015)
90. XILINX: Xilinx core generator. http://www.xilinx.com/tools/coregen.htm (2015)
91. XILINX: Xilinx dsps. http://www.xilinx.com/products/technology/dsp/ (2015)
92. Xilinx: Xilinx FPGAs, ISE. http://www.xilinx.com (2015)
93. XtremeData: XtremeData FPGA Boards. http://www.xtremedata.com/ (2015)
94. Zerbino, D.R., Birney, E.: Velvet: algorithms for de novo short read assembly using De Bruijn graphs. Genome Res. **18**(5), 821–829 (2008)
95. Zhang, W., Jha, N.K., Shang, L.: A hybrid nano/cmos dynamically reconfigurable system—part i: Architecture. J. Emerg. Technol. Comput. Syst. **5**(4), 16:1–16:30 (2009)

Springer

http://www.springer.com/978-981-10-0589-3

Architecture Exploration of FPGA Based Accelerators for BioInformatics Applications

Varma, B.S.C.; Paul, K.; Balakrishnan, M.

2016, XV, 122 p. 54 illus., 40 illus. in color., Hardcover
ISBN: 978-981-10-0589-3