Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2009, Article ID 518659, 15 pages
doi:10.1155/2009/518659

Research Article

SmartCell: An Energy Efficient Coarse-Grained Reconfigurable Architecture for Stream-Based Applications

Cao Liang and Xinming Huang

Department of Electrical and Computer Engineering, Worcester Polytechnic Institute, MA 01609, USA

Correspondence should be addressed to Xinming Huang, xhuang@ece.wpi.edu

Received 2 February 2009; Accepted 15 April 2009

Recommended by Markus Rupp

This paper presents SmartCell, a novel coarse-grained reconfigurable architecture, which tiles a large number of processor elements with reconfigurable interconnection fabrics on a single chip. SmartCell is able to provide high performance and energy efficient processing for stream-based applications. It can be configured to operate in various modes, such as SIMD, MIMD, and systolic array. This paper describes the SmartCell architecture design, including processing element, reconfigurable interconnection fabrics, instruction and control process, and configuration scheme. The SmartCell prototype with 64 PEs is implemented using 0.13 μm CMOS standard cell technology. The core area is about 8.5 mm^2, and the power consumption is about 1.6 mW/MHz. The performance is evaluated through a set of benchmark applications, and then compared with FPGA, ASIC, and two well-known reconfigurable architectures including RaPiD and Montium. The results show that the SmartCell can bridge the performance and flexibility gap between ASIC and FPGA. It is also about 8% and 69% more energy efficient than Montium and RaPiD systems for evaluated benchmarks. Meanwhile, SmartCell can achieve 4 and 2 times more throughput gains when comparing with Montium and RaPiD, respectively. It is concluded that SmartCell system is a promising reconfigurable and energy efficient architecture for stream processing.

1. Introduction

Nowadays, stream-based applications, such as multimedia, telecommunications, signal processing, and data encryptions, are the dominant workloads in many electronic systems. The real-time constraints of these applications, especially over portable devices, often have stringent energy and performance requirements. Many other military applications, including real-time synthetic aperture radar imaging, automatic target recognition, surveillance video processing, optical inspection, and cognitive radio systems, have similar needs. The general purpose solutions, such as general purpose processors (GPPs), are widely used in conventional data-path oriented applications due to their flexibility and ease of use. However, they cannot meet the increasing requirements on performance, cost, and energy in the data streaming application domain due to their sequential software executions. The application-specific integrated circuits (ASICs) become inevitably a customized solution to meet these ever-increasing demands for highly repetitive parallel computations. It is reported that they are potentially two to three orders of magnitude more efficient than the processors in terms of combined performances of computational power, energy consumption, and cost [1]. Although ASICs can provide best performance for specific applications, it is not desirable for all circuitry designs. ASICs generally have fixed data flow with predefined functionalities that makes them infeasible to accommodate to new system requirements or changes in standards. The long design cycle and high nonrecursive engineering (NRE) cost also become an obstacle to meet the stringent cost and time-to-market requirements.

Reconfigurable architectures (RAs) have long been proposed as a way to achieve a balance between flexibility as of GPP and performance as of ASICs. The hardware-based RA implementation is able to explore the spatial parallelism of the computing tasks in targeted applications, meanwhile avoiding the instruction fetching, decoding, and executing overhead of the software implementations, which results

in an energy and performance gain over general purpose processors. On the other hand, RAs maintain the postfabric flexibility to be configured, either offline or on the fly, to accommodate to new system requirements or protocol updates that is not feasible in ASIC implementations. Also, the flexibility provided by RAs can improve fault tolerance and system reliability. Design bugs can be easily fixed by loading new configurations, and malfunctioned circuitry can be excluded from other parts to achieve system recovery and prolong the product's lifetime.

Field-programmable gate arrays (FPGAs) are still the dominating semiconductor technology in the reconfigurable computing area. The most common SRAM-based FPGAs decompose complex logic functions into smaller ones and map them onto the Lookup Tables (LUTs) or other on-chip embedded resources. The island-style routing fabrics can be configured to form desired application datapath. The bit-level fine-grained granularity is suitable to implement a large variety of functions directly onto its rich hardware resources. However, this flexibility comes at a significant cost in terms of area, power consumption, and speed, due to its huge routing area overhead and timing penalty. Furthermore, due to the fine-grained nature, the compilation and configuration of FPGAs take much longer than those in general purpose processors.

In recognizing these issues, several research projects have been developed toward coarse-grained reconfigurable architectures, that include [2–7]. Benefiting from much less routing overhead, these coarse-grained reconfigurable architectures (CGRAs) have potential advantages to improve the power efficiency of the fine-grained FPGAs.

This paper presents SmartCell as a novel CGRA system, targeted for high-performance low-energy reconfigurable systems. SmartCell integrates a large number of tiny processor cores (cells) onto a single chip. The cells are interconnected with three levels of programmable switching fabrics, including intracell connection, nearest neighbor connection and a modified concentrated mesh (CMesh) on chip network. The feature of dynamic reconfigurability is achieved in two modes: coarse-grain cell broadcasting and fine-grain ID-based configurations. The number of processor elements involved in the computing tasks can also be dynamically changed to meet the application requirements. For example, more cells can be involved to achieve high computational performance, while fewer cells are put into active mode when power consumption is more stringent. It can be configured to operate in various computing styles such as SIMD, MIMD, and systolic array, targeted at applications with inherent data parallelism, high computing, and communication regularities.

The rest of this paper is organized as follows: after describing the background and examining some of the existing reconfigurable architectures in Section 2, the SmartCell architecture is detailed in Section 3, including the designs of computational units, layered interconnection fabrics, and control/configuration schemes. Section 4 presents a case study of mapping matrix multiplication onto the SmartCell and analyzing its performance. Section 5 presents the implementation of a seedling SmartCell system and its performance comparisons with other computing systems, followed by the conclusions in Section 6.

2. Background and Related Work

In recent years, the state of computing has evolved from a single CPU into a network of multiprocessors on chip running in parallel at relatively low frequencies. The major driving force behind is the need to achieve high computational capacity meanwhile maintaining high-energy efficiency. This feature has become critically important for many DSP applications, especially in portable devices. In this section, we briefly summarize several computing architectures targeted for data streaming applications.

Traditional FPGAs use the static configuration streams to control the functional and routing resources for user specifications. The data parallelism and flexible on-chip communications are essential to meet the high-performance requirements of the computations being performed. Due to the direct mapping of application tasks onto hardware resources, FPGA is able to complete one operation in a single clock cycle, which avoids the instruction fetching, decoding, and executing overheads as in the software processors.

However, as we mentioned previously, the fine-grained granularity and the general purpose LUTs in the FPGAs involve high routing overheads and intensive controlling requirements, which in turn degrades the performance and makes the compilation and configuration very slow. In realizing these problems, the commercial FPGA vendors have introduced more coarse-grained components as the computing basics in their newly developed FPGAs. The Virtex-4 [8] and Virtex-5 [9] series are among the latest Xilinx FPGAs, which have a mixed granularity of basic logic cells with coarse-grained DSP slices to enhance the signal processing capacity and power consumption performance. Similar idea can be found in Altera's Stratix II FPGAs [10]. In addition, 6-input and 8-input LUTs are introduced to the Virtex 5 and Stratix II FPGAs, respectively, as the substitute of the traditional 4-input LUT. This is helpful to reduce the routing overhead and to ease the configuration process. But the SRAM-based FPGAs have some fundamental limits that hamper them becoming the mainstream computing media for the data streaming applications. The system configuration SRAM cells are very greedy on power and area and are needed to be held during the entire operation process. It is studied in [11] that the FPGA consumes about 14 times more dynamic power and is about 35 times larger than equivalent ASICs on average when only logic elements are used. Furthermore, the FPGAs do not support instruction sequencing and are thus infeasible or very costly to make any changes on the fly. Unlike FPGAs, the SmartCell system can achieve the online flexibility by simply pointing to a new instruction code. The coarse-grained nature of the SmartCell avoids low level compilation and high routing overhead involved in the FPGA designs. The hardware circuitry for the domain specific operations also results in smaller area and better energy efficiency than FPGAs.

A number of researches have been carried out to explore efficient CGRA designs as summarized in [12]. The RAW [6] system incorporates 16 simplified 32-bit MIPS processors in a 2D mesh structure to provide high parallel computing capacities. The RaPiD [2] architecture links a large number of heterogeneous reconfigurable components in a 1D array structure, including ALUs, multipliers, and RAMs. The potential applications for RaPiD are those of a linear systolic nature or applications that can be easily pipelined among the computational units. In the PipeRench [4] system, several reconfigurable pipeline stripes are offered as an accelerator for data streaming applications. Limited configurable interconnection fabrics are developed, including a local network inside a stripe, unidirectional nearest neighbor connection between stripes, and some global buses. The Matrix [3] approach incorporates a large number of 8-bit Basic Functional Units (BFUs) in a 2D mesh structure. Its routing fabrics provide 3 levels of 8-bit bus connections, which can be configured into SIMD, MIMD, or VLIW styles. In the Chess [7] system, 4-bit ALUs are tiled in a hexagonal array with adequate reconfigurable interconnect fabrics to build a chessboard-like floorplan. The configuration contexts of a complex function can be cast and forwarded among active processors. Thus the functionality of the ALUs can be changed on a cycle-by-cycle basis. The MorphoSys [5] is an integrated and configurable system on chip, targeted for high throughput and data parallel applications. A modified RISC processor is embedded to control the reconfigurable accelerating arrays with efficient memory interface between them. The MorphoSys system is potentially to be operated in the SIMD style due to the column wise or row wise configuration broadcasts.

Many other reconfigurable architectures have been implemented with various technologies [13–15]. Most of them have been focused on the exploring of computational models or efficient design with respect to area and performance. The Processor-In-Memory- (PIM-) based systems [16, 17] integrate the processing logic and memories onto the same chip and try to perform the computations directly in memories, which greatly reduces data transfer overhead between CPU and main memory. The power consumption is another important aspect of reconfigurable architecture designs. In [18, 19], power efficient architectures are developed for specific applications and are compared with fine-grained implementations. However, they are not generic coarse-grained architectures but some specific models for data streaming applications.

Most recently, some other CGRA systems have also been developed to provide ultralow power consumption, such as ADRES [20] and Montium [21] with limited computing resources. The SmartCell system integrates some of the prominent features in the previous systems. The 16-bit granularity of the basic operations is efficient for the data parallelism exploration, while keeping a low cost of computing and communications. The SmartCell can also be configured to operate in SIMD, MIMD, and systolic array styles due to the distributed configuring contexts and rich on-chip connections. Dynamic configuration can be performed in both fine and coarse-grain modes. The uniform delay of the hierarchical interconnections also eases the scheduling of the stream processing among multiple cell units. In combination of these features, we say that the SmartCell system is a unique approach in the CGRA family. In Section 5, a direct comparison of the SmartCell performance in terms of energy efficiency and system throughput against RaPiD and Montium will be presented.

3. SmartCell Architecture

In this section, the proposed SmartCell architecture is described in detail, including the design of processing element, cell structure, on-chip interconnections, and system control and configuration schemes. Figure 1 depicts the components and organization of the integrated SmartCell architecture.

3.1. Key Features. In a typical SmartCell architecture, a set of cell units is organized in a tiled structure. Each cell block consists of four processing elements (PEs) along with the control and data switching fabrics. A three-level layered interconnection network is designed for the intra- and intercell communications. A serial peripheral interface (SPI) is designed as an efficient way to load/reconfigure instruction codes into active cell units. By reconfiguring the instruction memories, the data flow can be dynamically changed to accommodate to different application demands. Some important features of the SmartCell architecture are summarized as follows.

(i) Coarse-grained granularity: SmartCell is designed to generate coarse-grained configurable system targeted for computation intensive applications. The processing elements operate on 16-bit input signals and generate a 36-bit output signal, which avoids high overhead and ensures better performance compared with fine-grained architectures.

(ii) Flexibility: due to the rich computing and communication resources, versatile computing styles are feasible to be mapped onto the SmartCell architecture, including SIMD, MIMD, and 1D or 2D systolic array structures. This also expands the range of applications to be implemented.

(iii) Dynamic reconfiguration: by loading new instruction codes into the configuration memory through the SPI structure, new operations can be executed on the desired PEs without any interruption with others. The number of PEs involved in the application is also adjustable for different system requirements.

(iv) Fault tolerance: fault tolerance is an important feature to improve the production yields and to extend the device's lifetime. In the SmartCell system, defective cells, caused by manufacturing fault or malfunctioned circuits, can be easily turned off and isolated from the functional ones.

(v) Deep pipeline and parallelism: two levels of pipeline are achieved—the instruction level pipeline (ILP) in

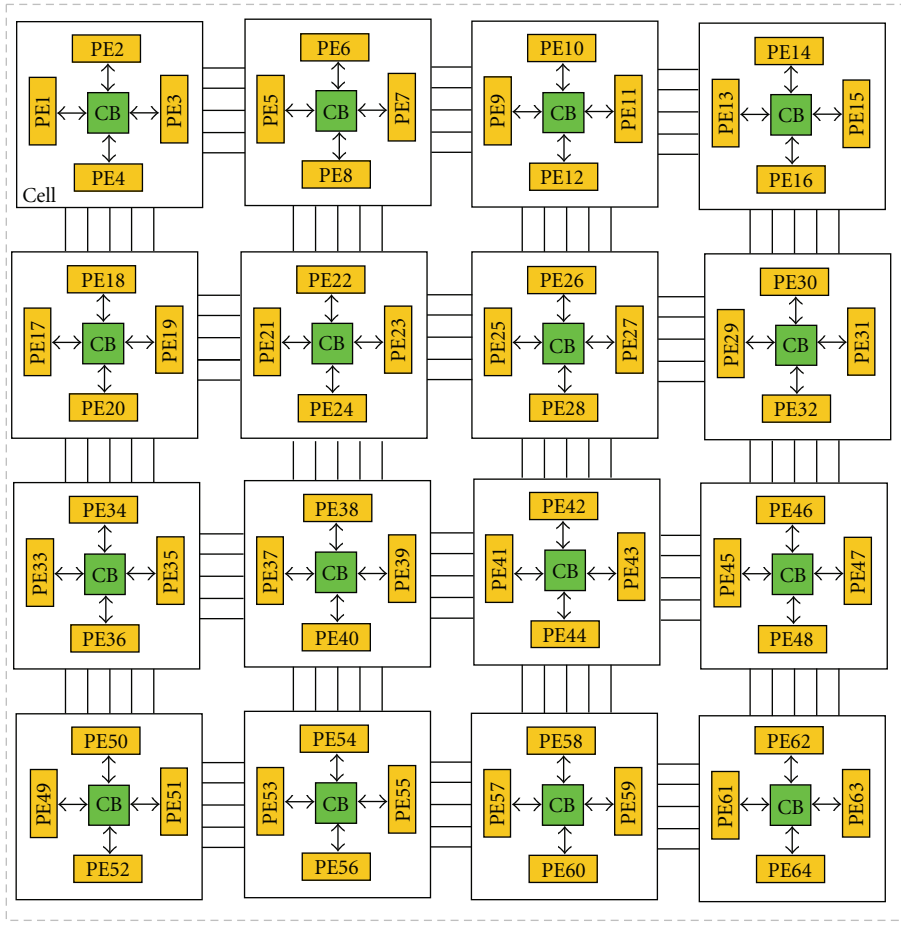

Figure 1: Overview of the SmartCell architecture. The SmartCell architecture is featured in a 2D tiled structure that consists of cell units and layered interconnection networks.

a single processor element and the task level pipeline (TLP) among multiple cells. The data parallelism can also be explored to concurrently execute multiple data streams, which in combine ensures a high computing capacity.

(vi) Hardware virtualization: in our design, distributed context memories are used to store the configuration signals for each PE. The cycle-by-cycle instruction execution supports hardware virtualization that is able to map large applications onto limited computing resources.

(vii) Explicit synchronization: a program counter (PC) is designed to schedule instruction execution time for each PE on the fly. Variant delays are also available for input/output signals inside each PE. Therefore, the SmartCell can provide explicit synchronization that eases the exploration of computing parallelisms.

(viii) Unique system topology: the cell units are tiled in a 2D mesh structure with four PEs inside each cell. This topology provides variant computing densities to meet different computational requirements. With the help of the hierarchical on-chip connections, the SmartCell architecture can be dynamically reconfigured to perform in variant operational styles.

3.2. Cell Unit and Processing Element. The reconfigurable cell units are the fundamental components in SmartCell, which are aligned in a 2D mesh structure as shown in Figure 1. Each cell consists of four identical PEs. The PE is composed of an arithmetic unit and a logic unit, I/O muxes, instruction controllers, local data registers, and instruction memories, as shown in Figure 2. It can be configured to perform basic logic, shift, and arithmetic functions. The arithmetic unit takes two 16-bit vectors as inputs for basic arithmetic functions to generate a 36-bit output without loss of precision during multiply-accumulate operations. The PE also includes some logic and shift operators, usually found in targeted data streaming applications. The basic operations supported by SmartCell processor are listed in Table 1. Multiple PEs can be chained together through the programmable on-chip connections to implement more complex algorithms.

An up to 4-stage pipeline structure is developed in each processor, as denoted in different colors in Figure 2. The *Src* select stage inputs data from the on-chip connection

Table 1: List of basic operations supported by SmartCell processors.

	Basic operations
Arithmetic Unit	add, sub, mult, MAC, abs sum
Logic Unit	and, or, not, xor, nand, compare, etc.
Shift Unit	shift right, shift left, circular shift

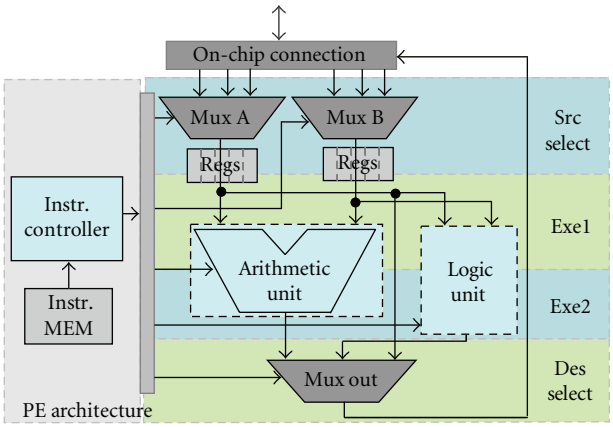

Figure 2: Processor element architecture. The PE component can be configured to perform 16-bit basic arithmetic operations, including logic, shift, adder, multiplier, and MAC. An instruction controller is designed for the cyclic configuration of the computational units and data flows.

calculated by other PEs or itself and stores the data into its local register banks. The execution stages (*Exe1* and *Exe2*) occupy two clock cycles for basic multiply-add and other logic operations. The *Des* select stage selects the output result and sends it back to the on-chip interconnections. Unlike traditional pipelined processor design, the pipeline stages are not fixed in SmartCell. Bypass path can be selected in every stage except for *Src* select to allow fast passing through input data or intermediate results to the next operating unit to reduce the processing delay if required. The traditional decoding stage is replaced by an instruction controller, which generates all control and scheduling signals in parallel with the 4 pipeline stages. An instruction code, pre-stored into the instruction memory, is loaded into the instruction controller on a cycle-by-cycle basis to provide both functionality and datapath control for a specific algorithm. In summary, the flexible 4-stage pipeline structure avoids the deep instruction pipelines of fetching, decoding, and registering read/write and ALU operations in conventional general purpose processors. Additionally, the instruction code can be dynamically reconfigured in various modes to adapt to different application requirements. Therefore, SmartCell is able to provide comparable energy efficiency as an ASIC while maintains dynamic programmability as a DSP.

The instruction code is designed in a 64-bit frame format, as shown in Table 2. A 9-bit program counter control (PC control) section is used to indicate execution time, next instruction address, and valid memory ranges for the mapped application. The datapath and operation

Table 2: Frame format of the instruction code.

	64 bits/instruction code					
No. of bits	9	20	7	10	11	7
Format	PC control	Datapath control	I/O delay	Operation control	NoC control	*RESV*

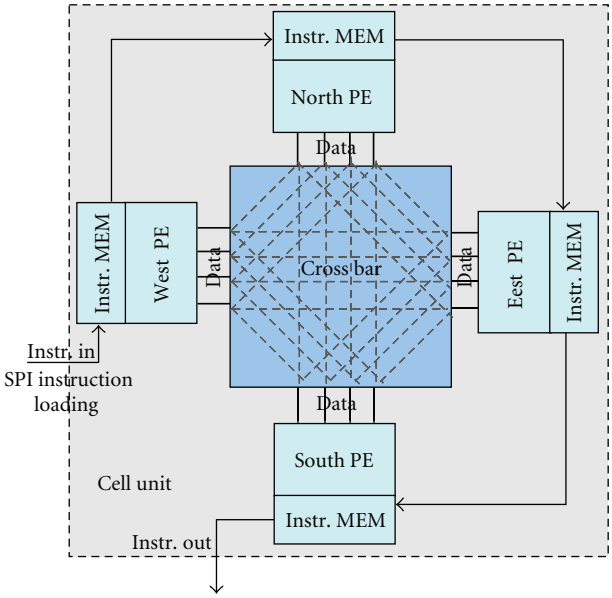

Figure 3: Cell unit structure. Each cell consists of 4 PEs and 4 instruction memories in pairs. A configurable crossbar is designed as the intracell data exchange network. The SPI buses are designed to form a ripple array structure for instruction loading and dynamic reconfiguration.

control signals specify the configuration of data flow and computing units, while the I/O delays are used for synchronization scheduling among multiple computing units. An 11-bit Network-on-Chip (NoC) control signal is designed to configure the on-chip communication network. A 7-bit undefined section is reserved for future functional extension. New instruction will not be loaded and executed until the current one expires. The instructions are accessed in a cyclic manner that supports periodical execution of a set of operations. In our current design, a 20 by 64-bit instruction memory block is attached to each PE.

In a cell unit, four PEs placed in the east, west, south, and north directions form a quad structure with a fully connected crossbar unit located at the center, as shown in Figure 3. The crossbar network supports arbitrary nonblocking connections among PEs in the same cell. Instruction memories are attached to each PE and are chained in a linear array fashion by serial peripheral interface (SPI) for configurations. The data exchange controls are also provided by the instruction code in a cyclic manner.

3.3. Three-Level Layered On-Chip Interconnection. As the CMOS technology scaling down, interconnect has become an increasingly important issue for integrated circuit design. In

many signal processing applications, the system throughput is significantly affected by communication costs. The design of efficient data exchange scheme has become a key feature for high-performance systems. Shared bus connection with high bandwidth is usually adopted in modern multicore CPU designs. But the lack of scalability and high power consumption penalty make it not favorable for data streaming applications. Other solutions are available for on-chip switch topology, such as fully connect crossbar and island-style mesh networks. The crossbar network provides the flexibility to connect any components in the network with limited transfer delays. Despite of these advantages, crossbar suffers from high silicon area costs, high power consumption, and low scalability. On the other hand, island-style mesh network is often used in FPGAs, in which each computing unit is attached with its own switch fabrics to transmit/receive data or to relay data to adjacent nodes. The mesh network offers regular structure and is easy to scale. But it suffers from longer delays and complex control logics. In realizing these facts, a compromised hierarchical routing structure with three-level networks is designed for the SmartCell: the fully connected crossbar unit for intracell data exchange, the static nearest neighbor connection for intercell communications, and the reconfigurable modified CMesh network for concurrent data communication among nonadjacent cell units.

3.3.1. Fully Connected Crossbar Intracell Interconnection. Initially, a centralized shared register memory (SRM) block is designed for the intracell communications. But it was abandoned due to its high area and power costs and complex memory access controls. In the current design, the PEs and instruction memories are placed at the four edges in a cell. A fully connected crossbar switch box is able to provide a nonblocking data exchange connection. Compared to the SRM implementation, the control logics are substantially simplified in the crossbar connection, which in turn results in a better timing and area performance.

3.3.2. Static Nearest Neighbor Intercell Interconnection. In our system, the homogeneous cell units are tiled in a 2D mesh structure. Thus the adjacent cells can be connected directly through short wires. Since four PEs in a cell are placed at four edges, each PE can be directly linked to the nearest PE located in the adjacent cell, as shown in Figure 1. This static network supports single cycle bidirectional data transmission of 2 16-bit and 1 36-bit signals between connected PEs. These signals are aligned up with the cell's internal signals to provide the PE's inputs. No extra synchronizations and delays are involved. This low latency and self-synchronization feature is critical for exploration of task level parallelism among cell units in many multimedia and signal processing applications.

3.3.3. Reconfigurable Hierarchical CMesh Network. Besides the local connections, it is realized that some applications also require dynamic data exchanges between nonadjacent cells, such as Radix-2 FFT, 2D DCT. After examining the major existing on-chip interconnection techniques, a modified CMesh network is adopted in our SmartCell architecture. It is studied in [22] that the CMesh network has the potential to provide best performance in terms of average latency and network efficiency among other NoCs, including Mesh, Torus, and FTree. As shown in Figure 4(a), the modified CMesh segments the network into clusters, with 4 cell units sharing the same switching component. The switch architecture, depicted in Figure 4(b), is designed to connect four local cells with adjacent cluster networks. Each cell inputs/outputs a 36-bit signal from/to the switch fabric that forms a local I/O interface. The switch component also receives two sets of 4 36-bit inputs from horizontal and vertical directions and outputs the same amount of data in these two directions. A routing arbitrator component is designed to dictate the proper data transmission that can be configured by the NoC control bits in the instruction code.

Dimension-Order Routing (DOR) is implemented to route data firstly in one direction and then in another for multihop data transmissions. Because no closed loop can be

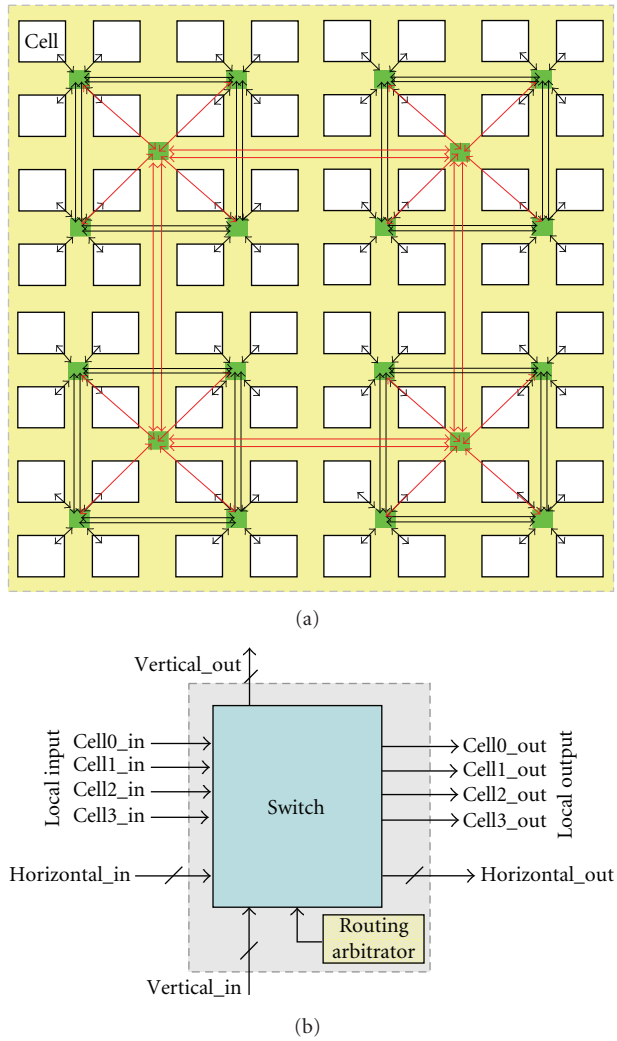

Figure 4: Reconfigurable hierarchical CMesh network: (a) modified CMesh Architecture; (b) switch fabrics of the CMesh network.

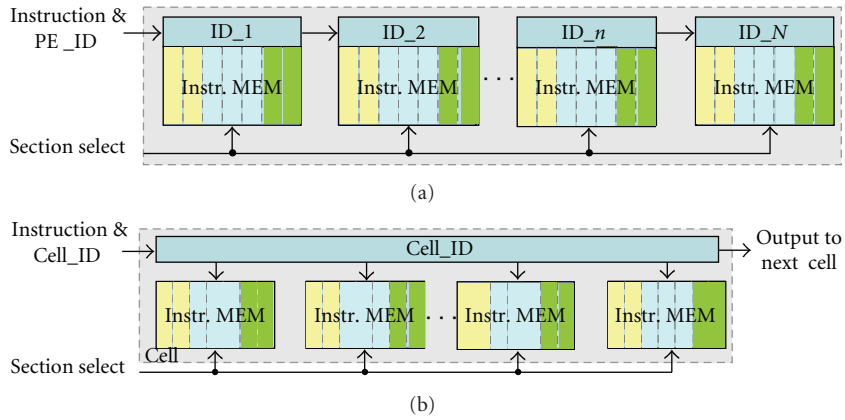

Figure 5: Diagram of two modes of reconfigurations. (a) ID-based fine-grain configuration. (b) Cell broadcasting coarse-grain configuration. A global select signal is developed for memory partitioning.

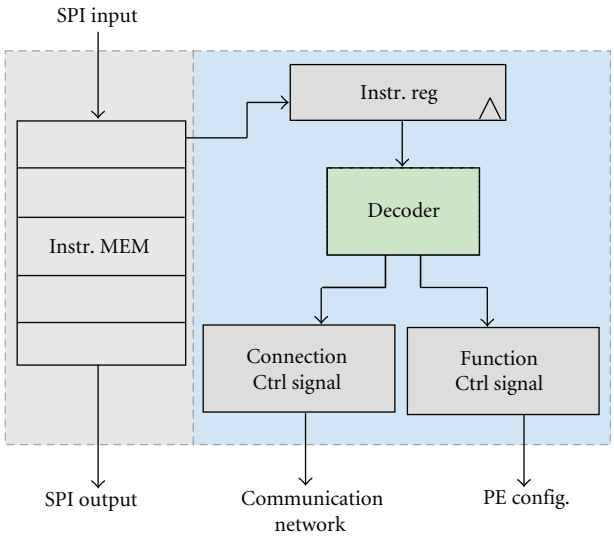

Figure 6: Diagram of configuration controller architecture. The control signals for the PE functionality and the data communications are decoded based on the current instruction code stored in the instruction register. The SPI interface is designed for instruction loading and updating.

generated, the DOR scheme guarantees no traffic contention exists. Instead of arranging the routers in a ring style as in traditional CMesh network, high-level routers that connect four local routers are designed and chained together to form another CMesh network. It is so-called hierarchical CMesh network. This hierarchical CMesh network reduces the number of long wires compared with traditional CMesh. Also, the same routing components can be easily added to or reduced from the SmartCell architecture for system scalability. In our design, each cell can receive a 36-bit signal through the CMesh network every clock cycle, which leads to a single-hop system throughput of 57.6 Gbits/s for a 4 by 4 SmartCell operating at 100 MHz.

3.4. Configuration and Control Flow. A serial peripheral interface (SPI) is designed to configure and update the instruction memories, as shown in Figure 3. In this structure, the instruction memories are linked in a ripple array fashion with the inputs and outputs chained one to another. During the initial configuration procedure, the instruction code is loaded to the first PE's instruction memory and is then shifted down to the second one and so on. This procedure stops after the last active PE is configured. The run-time reconfiguration can be achieved by the same SPI structure. Two modes are provided for the fine-grained ID-based configuration and coarse-grained broadcasting, as shown in Figures 5(a) and 5(b). Some applications require fine control of individual PE to perform different tasks. The ID-based fine-grained configuration is used in this case. The new instruction code and the ID of the PE to be configured are sent into the SPI chain. The PE bypasses the information to the next one until it reaches the desired PE. On the other hand, a group of PEs is configured to perform the same operation in the SIMD style for many other applications. To reduce latency, a cell broadcasting coarse-grained configuration is designed to currently write the reconfiguring contexts into all instruction memories in the same cell, based on the input Cell ID. In a 4 by 4 SmartCell system, 32 and 8 clock cycles are needed on average for an instruction code to reach the configuration component in fine-grain and coarse-grain modes, respectively.

However, the configuring propagation latency is not the same for different PE/Cell units along the SPI chain: the nearer to the input port, the faster the configuration can be done. To compromise this unbalanced configuration latency, a memory partitioning scheme is developed in our design. In this scheme, new instruction codes can be loaded into the unused context memories while the PEs are still operating in the current contexts. Once the new contexts are fully loaded, a global select signal is used to indicate the switch of operation context. The configuration latency is hidden by this means. Multiple contexts can be efficiently swapped within one clock cycle.

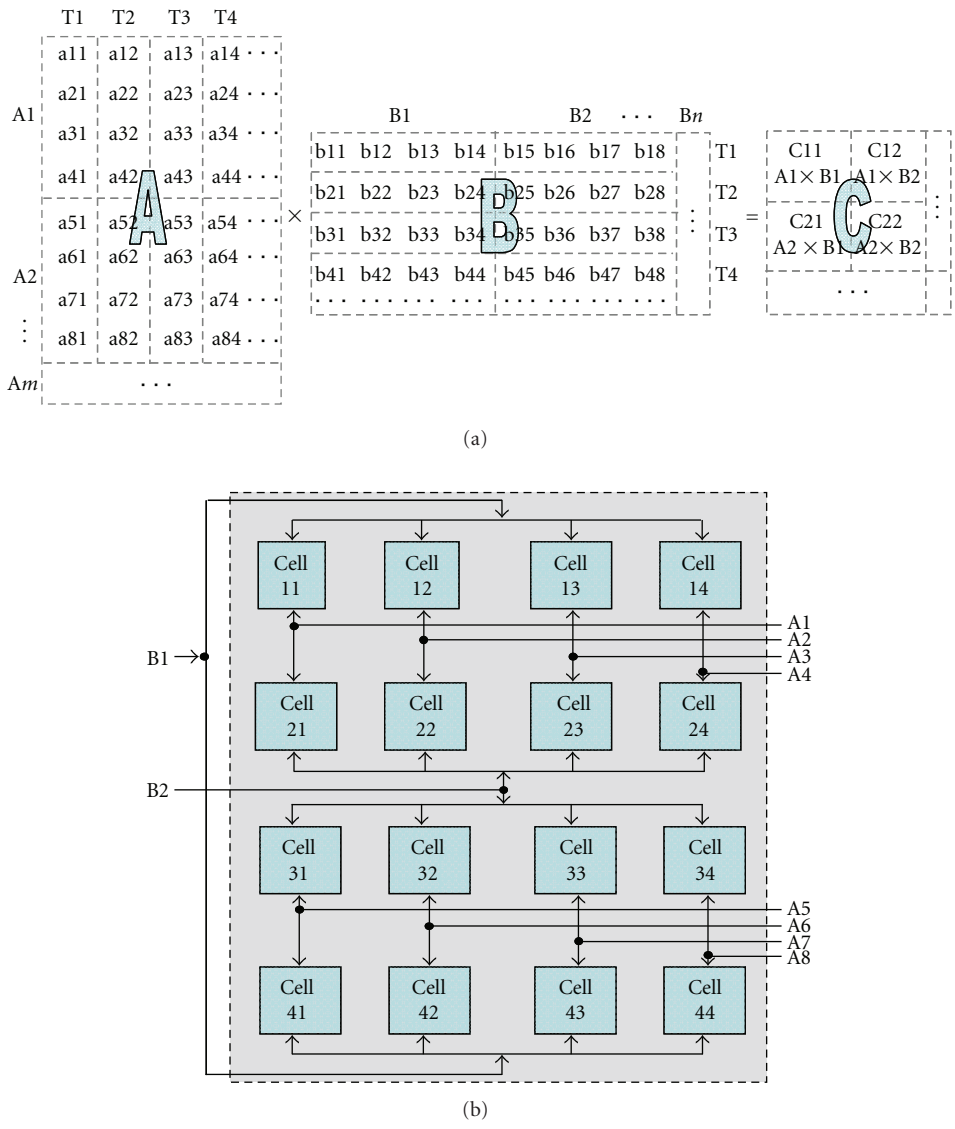

FIGURE 7: Illustration of subblock matrix multiplication algorithm and mapping onto SmartCell. (a) Subblock matrix multiplication scheme with timing information; (b) algorithm mapping and data sharing on SmartCell system.

Pipline structure of 4 PEs in the smae cell

	Time	Pipe 1 PE	Pipe 1 OP	Pipe 2 PE	Pipe 2 OP	Pipe 3 PE	Pipe 3 OP	Pipe 4 PE	Pipe 4 OP
T1	1	1	c11 = a11 × b11						
	2	2	c12 = a11 × b12	1	c21 = a21 × b11				
	3	3	c13 = a11 × b13	2	c22 = a21 × b12	1	c31 = a31 × b11		
	4	4	c14 = a11 × b14	3	c23 = a21 × b13	2	c32 = a31 × b12	1	c41 = a41 × b11
T2	5	1	c11+ = a12 × b21	4	c24 = a21 × b14	3	c33 = a31 × b13	2	c42= a41 × b12
	6	2	c12+ = a12 × b22	1	c21+ = a22 × b21	4	c34 = a31 × b14	3	c43 = a41 × b13
	7	3	c13+ = a12 × b23	2	c22+ = a22 × b22	1	c31+ = a32 × b21	4	c44 = a41 × b14
	8	4	c14+ = a12 × b24	3	c23+ = a22 × b23	2	c32+ = a32 × b22	1	c44+ = a42 × b21
	...								

FIGURE 8: Pipelined computations for one subblock result of matrix C. The data in red circle denotes the external inputs in each time step.

Figure 6 depicts the control flow in a processing unit. At run-time, a configuration context is loaded into the instruction register and is then partitioned into interconnection and functionality controls. The next context is loaded only after the current one expires. Cyclic data flows can be configured through the looping of instructions in the memory.

4. A Case Study: Mapping of Matrix Multiplication onto SmartCell

A broad range of complex scientific and multimedia applications strongly depends on the performance of matrix-matrix multiplication. In this section, we use matrix multiplication as a case study to demonstrate how to map applications onto the SmartCell and to analyze its performance.

Various methods have been proposed in literature for high-performance matrix multiplication designs, such as Cannon's algorithm [23], Strassen's algorithm [24], and more recently systolic algorithms using special systolic arrays. A subblock matrix multiplication scheme is adopted in our design. In this scheme, the operand matrices are partitioned into smaller submatrices, each of which is then processed separately by different hardware resources in parallel. The result matrix is generated into subblocks of regular dense matrix. This scheme can be efficiently mapped onto our SmartCell system to explore both spatial and temporal parallelisms for high computing performance. At the same time, it achieves good data reusability among hardware resources to ease the stress of external memory bandwidth requirement.

In our design, the operand matrices A and B are partitioned into subblocks of 4 rows and 4 columns respectively, as shown in Figure 7(a). They are then fed to the computing resources in column-major and row-major order, respectively, with the timing sequences from T1 to T4 depicted in the same figure. The independent 4 by 4 subblock results can be potentially calculated in parallel by different computing resources. Given the SmartCell architecture, each 4-row/column pair can be efficiently mapped onto the 4 PEs in the same cell unit. The mapping of the subblock matrix multiplication onto a 4 by 4 SmartCell system is illustrated in Figure 7(b). In this scheme, eight subblocks of matrix A are concurrently input to the cell units and each subblock is shared between two vertically placed cells. Two subblocks from B matrix are simultaneously broadcast to the rows of cells with one block shared by two rows. By this means, 16 subblocks of the result matrix C can be computed in parallel. For further performance optimization, inner cell pipeline structure is also explored. The pipeline structure of one subblock result is illustrated in Figure 8. Initially, element pair (a11, b11) is loaded into PE1 and generates partial result of c11. The calculated result is stored in the local register that can be accumulated during the next loops. After that, the second pair (a21, b12) is loaded into PE1 and PE2 along with previous data for the computation of c21 and c12, respectively. The calculations of the 4 rows in the result matrix are carried out in 4 pipes. Data used by the previous pipe is shared by the next pipe during the following time step through the crossbar unit. After 4 time steps, all pipes are filled up with computing data and are able to operate at full rate. This scheme only requires input of two external data in each step to maintain full operation of 4 PEs as highlighted in red circles in Figure 8.

To evaluate its performance, a 32 by 32 square matrix multiplication is mapped onto a 4 by 4 SmartCell system. In general, each final element requires 32 clock cycles to finish the 32 MAC operations involved in it. Due to the fully pipelined structure, a 4 by 4 subblock result can be calculated in a single cell within 128 clock cycles. The final 32 by 32 matrix C is decomposed into 64 independent 4 by 4 subblocks, which can be calculated by the available 16 cells in parallel. Thus a total number of 512 clock cycles is needed to compute a single 32 by 32 matrix multiplication, which leads to a system throughput of 195.3 KMatrices/s running at 100 MHz.

Table 3: Application domain and test benches.

Application domain	Test benches
Signal processing	64-tap FIR, 32-tap IIR
Multimedia and image processing	64-point FFT, 8 by 8 2D-DCT, 8 by 8 Motion Estimation (ME) in 24 by 24 area
Scientific computing	128 by 128 MMM, 64th order Polynomial Evaluation (PoE)

5. Hardware Synthesis Results and Performance Comparisons

In this section, a SmartCell system with 4 by 4 cell units is designed and synthesized in standard cell ASICs. The area and timing performance is provided based on the synthesis reports. Seven benchmark applications listed in Table 3 have been manually mapped onto the SmartCell system. These benchmarks represent a wide range of real-time applications from signal/image processing to scientific computing. The power/energy consumption and system throughput results are then compared with other computing platforms, including FPGA and standard cell ASICs. Finally, we compare the energy efficiency and system throughput performance with other CGRA systems, including RaPiD and Montium. The reason to choose RaPiD is that it shares similar hardware resources with our design, and the performance of interest for a common set of benchmarks are disclosed in details. Montium is among several recently developed ultralow power CGRA systems. It uses the same process technology as SmartCell. Its performance results for some benchmarks are also available in literature. The system throughputs and energy consumption are compared among these systems. The software programming environment is also presented at the end of this section.

5.1. SmartCell Prototype. The prototype SmartCell system is developed and synthesized with standard CAD tools.

Table 4: System design and simulation parameters.

System dimension	4 by 4
Design tools	ModelSim, Synopsys CAD tools
Library	TSMC 0.13 μm process
Synthesis environment	Worst case condition
Voltage	1 V
Simulation frequency	100 MHz

A functional RTL model is firstly designed in HDL hardware description language and is then synthesized in Synopsys DesignCompiler to generate the CMOS standard cell ASICs using TSMC .13 μm technology. No custom optimization is applied during this process. The area and timing results are also generated by DesignCompiler using worst case conditions. The synthesized design is then annotated via a set of benchmarks for power consumption estimations in Synopsys PowerCompiler. Some experimental setups are listed in Table 4.

For power consumption and system throughput evaluations, all benchmarks are simulated at the same operating frequency of 100 MHz. The same simulation frequency was also used by RaPiD for its power consumption analysis. Because the RaPiD was designed in 0.5 μm process and was operated at 3.3 V, a fair comparison requires scaling down its power consumption by a reasonable factor. In our study, full scaling [25] is performed that scales down the power consumption from 3.3 V to 1 V by a factor of 3.3^2. By this means, the process dimension is also scaled down to 0.15 μm. To compensate the effect of dimension scaling, constant voltage scaling [25] is then performed to scale up the power consumption by a factor of 1.7. Therefore, the RaPiD power consumption is scaled down by an overall factor of 9.34.

The same benchmarks are also directly implemented on the FPGA platform to provide performance comparison. The state-of-the-art Altera's Stratix II FPGA in 90 nm process technology is selected as the benchmark platform. In particular, an EP2S30 FPGA device is used, since it is the smallest Stratix II FPGA that contains the same number of multipliers as in the SmartCell system. The benchmarks are designed in Altera's Quartus II 6.1 CAD tool and simulated at 100 MHz in ModelSim. The PowerPlay Analyzer is used to evaluate the power consumption based on the switching annotation generated from the gate level simulations. For fair comparison, only the core power consumption is recorded in the FPGA implementation, since the I/O and aux power is not included in others.

The ASIC implement is also generated for each test bench using the same HDL code as in the FPGA designs. It is expected to provide the best power/energy performance at a cost of flexibility. We use the same 0.13 μm process technology as in the SmartCell. Due to the large set of benchmarks under test, standard cell circuits are generated automatically by the Synopsys CAD tools without custom optimizations. We estimate the power consumption of the ASICs based on the gate level simulations at 100 MHz.

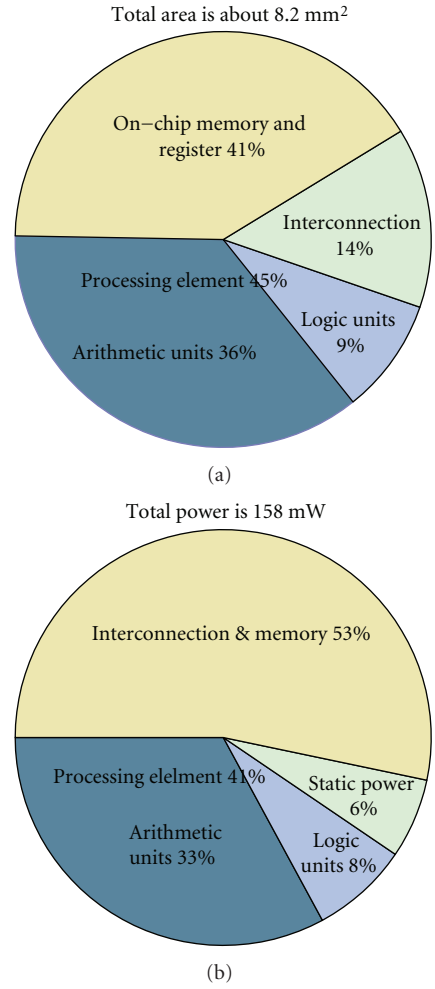

Figure 9: Area and average power consumption of the 4 by 4 SmartCell system: (a) area breakdown, (b) average power consumption breakdown at 100 MHz.

Similarly, only the logic core power is recorded as done in the other platforms.

5.2. SmartCell Area, Timing, and Power Consumption Performance. The area of the SmartCell system according to the synthesis tool is about 8.2 mm^2, which is about 1.6 million gates. The system area, breakdown into PE, on-chip memory and registers, and interconnection, are shown in Figure 9(a). The area of the processing elements is further decomposed into arithmetic units and logic units. The interconnection area is calculated by subtracting the area of the processing units and on-chip memories from the total area. The synthesis results indicate that the processing units contribute to about 45% of the total area, with 36% for arithmetic units and 9% for logic units. The on-chip memory and register together comprise about 41% of the area, mainly due to the long instruction format and intensive controlling requirements. Furthermore, custom optimizations or library components are likely to reduce the on-chip

memory occupation area. The three-level hierarchical on-chip interconnections roughly take 14% of the total area.

The SmartCell can operate at a maximum frequency of about 123 MHz. Further investigation reveals that the single cycle MAC unit inside the arithmetic component takes about 5.5 nanoseconds of the total critical path delay, with 3.2 nanoseconds for the 18-bit multiplier and 2.3 nanoseconds for the 36-bit adder. Again, custom optimizations can be performed to improve the timing result such as using pipelined multiplier and carry lookahead adder. Also the 123 MHz maximum frequency is captured from the worst case critical path delay. Register delay is available between the MULT and ADD components to break the MAC operation into 2 cycles. Therefore, the critical path that is used for most benchmarks is shorter than that reported by the CAD tools. The configuration time is another important matrix in reconfigurable architecture designs. In SmartCell system, the full chip fine-grain configuration can be completed within 13 microseconds at 100 MHz. Dynamic reconfiguration can be much faster, depending on how different the new configuring context is. For example, 64 cycles are needed to reconfigure the SmartCell from 2D-DCT to 64-tap IIR applications. Furthermore, if both configuring contexts have already been preloaded into the instruction memory, only one cycle is required to switch between IIR and DCT applications using the memory partitioning scheme. For most applications under test, the SmartCell can be dynamically reconfigured in less than 1 microsecond, which is much faster than the fine-grained FPGA reconfigurations.

The power consumption of the SmartCell for different benchmarks is estimated in PowerCompiler based on netlist annotation from gate level simulations. Table 5 lists the power consumption (dynamic power P_{Dyn} and total core power P_{Core}) and energy efficiency (E_{Eff}) performance for seven benchmark applications. All figures are generated at 100 MHz. Clock gating is automatically added by synthesis tool to dynamically turn off the clocks for unused registers. This requires an enable signal, indicating whether the register is in use or not, to be attached to each register during the design stage. Figure 9(b) shows the power dissipation for different parts of the SmartCell system. The processing units consume about 41% of total power, with 33% for ALUs and 8% for logic components. The on-chip memories and interconnections consume another 53% of total power. On average, the SmartCell consumes about 160 mW at 100 MHz. At last, the energy efficiency, evaluated by the total number of operations per second per watt, is also calculated, as shown in Table 5. A peek performance of 44.1 GOPS/W is achieved in the motion estimation application. SmartCell provides an average 37.8 GOPS/W energy efficiency from all benchmarks under test.

5.3. Comparison of Power/Energy Consumption with FPGA and ASICs. In this section, we compare the power and energy consumption performance with other computing platforms, including FPGA and ASIC. Table 6 gives the power consumption and system throughput of each benchmark, all generated at 100 MHz. Due to similar algorithm mapping structures and the same simulating frequency, the evaluated platforms can achieve the same system throughput for all benchmarks except for 64-point FFT. Pipelined FFT structure is adopted in both FPGA and ASIC implementations, which generates 1 output per clock cycle after some initialization time. While in our design, a parallel structure is mapped onto the SmartCell system, with 16 butterfly units running concurrently. Consequently, 60 clock cycles are required in the SmartCell to complete 1 block of 64-point FFT, which yields a throughput of 107 MS/s. Figure 10(a) compares the power consumption results of these three platforms, which has been normalized to the results of ASIC implementations. As expected, the ASIC implementations outperform both SmartCell and FPGA. SmartCell is about $2.7 \sim 5.4$ less power efficient than ASICs. The maximum gap is observed in the FFT applications. A reason for this is that a smaller number of multipliers are used in the pipelined FFT structure and fewer memory data switching activities are involved. On the other hand, SmartCell outperforms FPGA by a factor of $2.7 \sim 4.8$ in terms of power consumption.

Table 5: SmartCell power consumption and energy efficiency of different benchmarks at 100 MHz.

	FIR	IIR	MMM	2D DCT	ME	FFT	PoE
P_{Dyn} (mW)	143	180	137	156	142	165	141
P_{Core} (mW)	152	189	146	165	151	174	150
E_{Eff} (GOPS/W)	42.1	33.9	44.8	38.8	44.1	18.3	42.7

A more meaningful figure is depicted in Figure 10(b) that compares the average energy efficiency (GOPS/W) among the evaluated platforms, normalized to FPGA result. As expected, the ASICs are the most energy efficient among the evaluated platforms. It provides an average 16.4x energy efficiency gain compared with the FPGA result. However, this performance gain is achieved at a cost of no postfabrication flexibility and high engineering design cost. The energy performance provided by the SmartCell falls somewhere in between. It is about 4.1x more energy efficient than FPGA and is about 4x less than the ASIC implementations. This result demonstrates that the coarse-grained architecture is able to fill the energy efficiency gap between the fine-grained FPGA and logic specific ASIC architectures.

5.4. Comparison with Other CGRA Systems. In this section, we compare the SmartCell system with some other CGRA systems, including Montium and Rapid. SmartCell and Montium [26] occupy about 8.2 mm^2 and 1.8 mm^2 silicon area, respectively, with the same 0.13 μm technology, while RaPid [27] consumes about 5.7 mm^2 in 0.5 μm technology. The power consumption of different benchmarks is given in Table 7. On average, Montium consumes 3.2x and 7.5x less power than SmartCell and RaPid, respectively. However, the direct comparison of power consumption does not mean much, due to different system configurations, hardware resources, computing precision, memory sizes, and so forth. For the same reason, the amount of actual operations per second cannot be easily generated to compare the

Table 6: SmartCell power consumption and energy efficiency of different benchmarks at 100 MHz.

	SmartCell		FPGA		ASIC	
	Power (mW)	Throughput	Power (mW)	Throughput	Power (mW)	Throughput
FIR	152	100 MS/s	725	100 MS/s	31	100 MS/s
IIR	189	100 MS/s	896	100 MS/s	45	100 MS/s
MMM	143	763 Metrics/s	445	763 Metrics/s	36	763 Metrics/s
2D-DCT	165	2.8 MBlocks/s	795	2.8 MBlocks/s	60	2.8 Mblocks/s
ME	145	3.5 MBlocks/s	573	3.5 MBlocks/s	27	3.5 Mblocks/s
FFT	174	107 MS/s	475	100 MS/s	32	100 MS/s
PoE	150	100 MS/s	628	100 MS/s	55	100 MS/s
RC5	140	50 MBlocks/s	553	50 MBlocks/s	9	50 MBlocks/s

energy efficiency as calculated in Section 5.3. Instead, a more realistic way is to compare the total energy consumption to finish the same amount of tasks. It provides a fair comparison of the relative energy efficiency among evaluated systems. The system throughput is also calculated and compared based on the number of clock cycles required to finish these tasks.

As listed in Table 7, five benchmarks have been mapped onto the SmartCell system. Three of them are shared by the RaPid and Montium, individually. Montium achieves the best power consumption performance, since limited computing resources of only 5 ALUs are provided. The cycle column in the table denotes the number of clock cycles needed to compute one data block, except for the FIR filter design. In the 20-tap FIR benchmark, 2 blocks of 512 samples are used to generate the cycle and energy figures, as done in [26]. The results demonstrate that in most applications, the SmartCell requires less clock cycles to finish the same amount of task comparing with the RaPid and Montium implementations. This is benefitted from more processing parallelism provided by SmartCell to reduce the computing complexity. For example, in the SmartCell implementation, three data pipes can be created to process the 20-tap FIR filter in parallel. On the other hand, a recursive processing scheme is adopted in Montium, since at most 5-tap FIR can be handled at the same time. This recursive scheme also involves extra control and data exchange overheads. The energy consumption is also compared in Table 7, which is computed as the product of average power consumption and number of clock cycles.

The normalized energy consumption is depicted in Figure 11(a). When the Montium system is compared, the SmartCell dissipates the same amount of energy in the 64-point FFT and consumes about 18.6% more energy in the 8 by 8 2D DCT. For the 20-tap FIR benchmark, 42.0% energy saving is observed in our SmartCell implementation. On the other hand, SmartCell always outperforms RaPid with respect to energy consumption. A maximum 11.7 energy gain is achieved in the 128 by 128 matrix multiplication. On average, SmartCell is about 7.8% and 69.3% more energy efficient than Montium and RaPid, respectively, for evaluated benchmarks.

Figure 11(b) compares the normalized system throughput of different platforms. The SmartCell and the RaPid provide same throughput in motion estimation application, due to similar algorithm mapping structures. SmartCell outperforms both Montium and RaPid in all other benchmarks regarding to system throughput. In the FIR application, the SmartCell is about 6x faster than Montium system. SmartCell also shows a maximum throughput gain of 4.2x over RaPid system in the matrix multiplication implementations. Averagely, SmartCell provides about 4.0x and 2.2x throughput gains against the Montium and RaPid systems, respectively.

5.5. Development of Software Environment. Another important aspect in our research is to develop a software programming environment to assist the automatic application mapping onto the SmartCell system. Due to the control intensive nature, a prototype software compiler, named *Smart_C*, is proposed to facilitate the context generation in targeted application domain. Figure 12 represents the generic application mapping flow of the *Smart_C* environment.

Two phases are included in this design flow: application and architecture analysis phase (Phase I) and application mapping phase (Phase II). During Phase I, a high-level application description file (preferably in C language) is input into the *Smart_C* environment. An application abstraction step is performed to parse the input application file and to extract the work loads from it. All candidate loops are broken into linear sequences. The data dependencies among them are also analyzed during this step. On the other side, a hardware description and system requirement file are input to generate the hardware abstract, which specifies the computing resources and IO models for all available PEs. At last, the parallelism/pipeline exploration and application partitioning are performed to create scheduling code based on the hardware and software abstracts. The communication flow among active PEs are also scheduled here. The second application mapping phase transforms the scheduling code into configuring contexts that can be directly loaded into the instruction memories. Firstly, the control signals are determined for every computing and communication component to form the desired application datapath. Two modes are

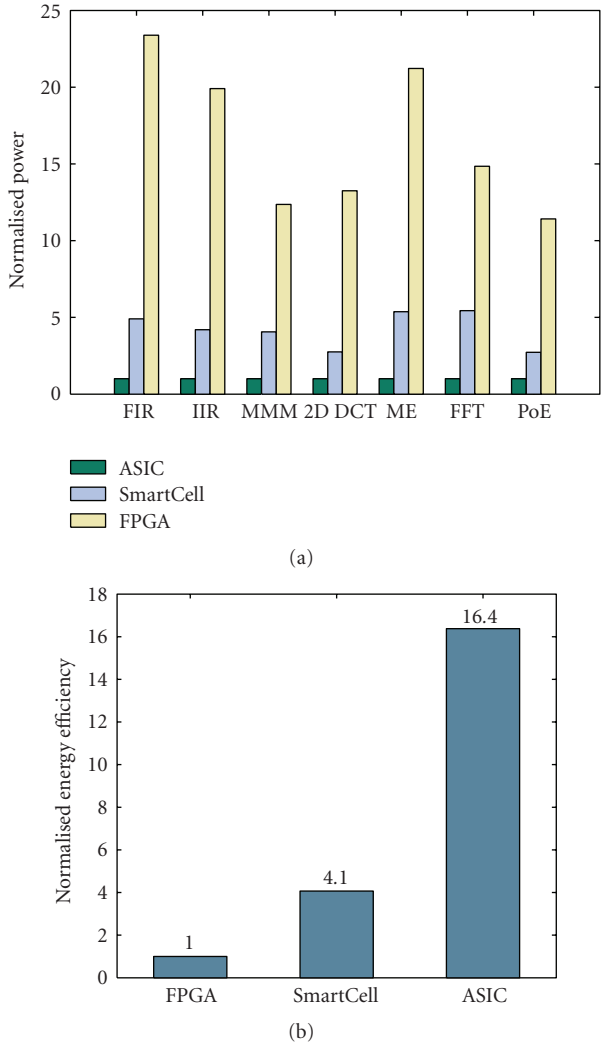

Figure 10: Diagram of power and energy efficiency comparisons among SmartCell, FPGA, and ASICs. (a) Power consumption for different benchmarks, normalized to the ASIC results; (b) average energy efficiency comparison, normalized to FPGA result.

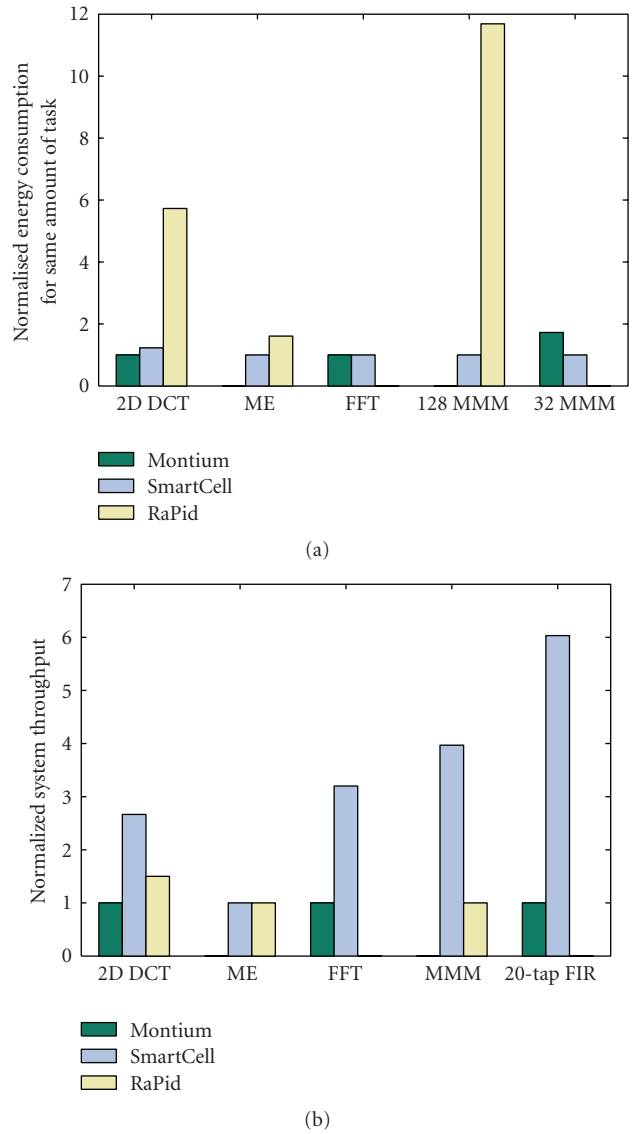

Figure 11: Diagram of energy consumption and throughput comparison among Montium, SmartCell, and RapiD. (a) Normalized energy consumption for same amount of task; (b) normalized system throughput.

provided for offline and online configurations. In the offline configuration, the context file can be directly downloaded into the instruction memories for all active PEs. On the other hand, if the online configuration is performed, the differences between the current context and the generated one are examined. Only the PEs observing different contexts are needed to be updated.

At the current stage, the application mapping environment (Phase II) has been implemented to generate the configuration contexts from an input assembly code. Due to the regular system structure and data flow pattern involved in the targeted application domain, the cell level configuration can be generalized to use the same prototype models. In this case, we create two sets of computation and communication libraries, each of which specifies all available computing operations and I/O models for a cell unit. Given an application, the designer is responsible to properly partition the kernel operations onto the PE components and to explore the data flows among them. After that, an application description code can be created in representing of the computing and I/O models specified in the assembly libraries. According to these models, the context generator is able to automatically decode the control signals for the computing components from the input assembly code. The interconnection context can also be generated based on the input I/O models and the system architecture configuration. By this means, the configuration overhead can be greatly reduced. The following steps remain the same as described earlier. The development of the system analysis phase (phase I) involves lots of experiences on system and task level profiling, analysis, and optimizations as usually found in complex compiler designs. Benefitting from the regular tile structure

Table 7: Power and energy comparison among the evaluated CGRA systems.

	SmartCell			RaPid [27]			Montium [21, 26]		
	Power (mW/MHz)	Cycles	Energy (nJ)	Power (mW/MHz)	Cycles	Energy (nJ)	Power (mW/MHz)	Cycles	Energy (nJ)
2D DCT	1.65	36	59	4.29	64	275	0.5	96	48
ME	1.46	1156	1688	2.35	1156	2717	—	—	—
FFT	1.74	60	104	—	—	—	0.541	192	104
MMM	1.46	33 K	48 K	4.28	131 K	561 K	—	—	—
20-tap FIR	1.47	341	501	—	—	—	0.42	2057	864

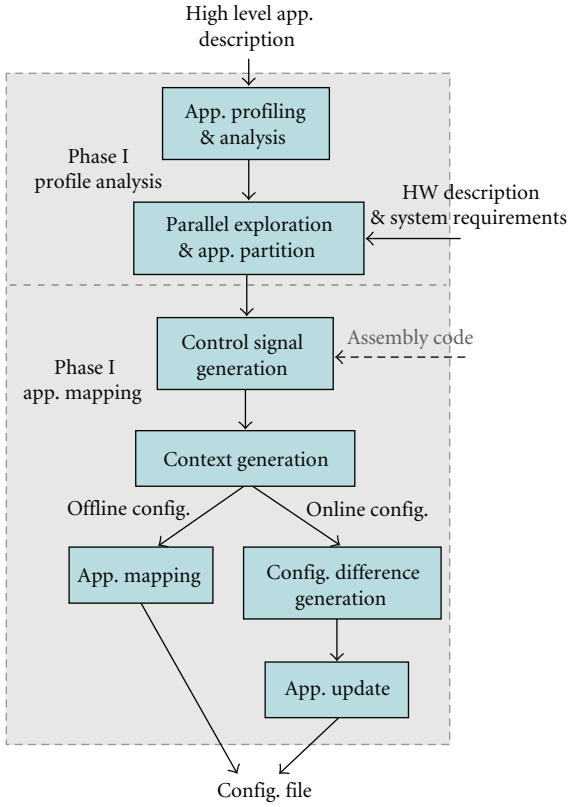

Figure 12: Design flow and application mapping environment for SmartCell.

and uniformed control logics, SmartCell can be configured for different system requirements of high performance or ultralow power consumption. The compiler needs to be robust enough to take advantage of hardware flexibility. One possible solution is that the compiler can read in a system constraint file, providing the system requirements, available hardware resources, targeted frequency, and so forth, based on which the configuration contexts are generated to satisfy these system requirements. Hardware virtualization also needs to be handled by the compiler to breakdown large computing tasks into smaller ones that can be fitted on the hardware resources and be processed individually. The optimization of task partitioning and scheduling needs to be properly addressed in the compiler design to explore both spatial and temporal parallelism potentially offered by SmartCell. Other issues such as loop breaking, redundancy optimization also need to be addressed in the compiler algorithm design. These challenges lead to couple interesting directions for our future work.

6. Conclusions

This paper presents the SmartCell as a novel reconfigurable architecture for stream-based applications. It is a coarse-grained architecture that tiles a large number of processor elements with reconfigurable communication fabrics. A prototype with 64 PEs is implemented with TSMC 0.13 μm technology. This chip consists of about 1.6 million gates with an average power consumption of 1.6 mW/MHz for the evaluated benchmarks. The benchmarking results show that SmartCell is able to bridge the energy efficiency gap between the fine-grained FGPAs and customized ASICs. When compared with Montium and RaPid, SmartCell shows 4x and 2x throughput gains and is about 8% and 69% more energy efficient, respectively. The performance results show that SmartCell is a promising reconfigurable and energy efficient platform for stream processing.

Acknowledgments

This work has been supported in part by the Defense Advanced Research Projects Agency (DARPA) Young Faculty Award under Grant W911NF-07-1-0191-P00001, and by the National Science Foundation (NSF) through award ECS-0725522.

References

[1] B. Khailany, W. J. Dally, U. J. Kapasi, et al., "Imagine: media processing with streams," *IEEE Micro*, vol. 21, no. 2, pp. 35–46, 2001.

[2] C. Fisher, K. Rennie, G. Xing, et al., "Emulator for exploring RaPiD configurable computing architectures," in *Proceedings of the 11th International Conference on Field-Programmable Logic and Applications*, pp. 17–26, 2001.

[3] E. Mirsky and A. DeHon, "MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources," in *Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 157–166, 1996.

[4] H. Schmit, D. Whelihan, A. Tsai, M. Moe, B. Levine, and R. R. Taylor, "PipeRench: a virtualized programmable datapath

in 0.18 micron technology," in *Proceedings of the Custom Integrated Circuits Conference*, pp. 63–66, 2002.

[5] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. C. Filho, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Transactions on Computers*, vol. 49, no. 5, pp. 465–481, 2000.

[6] M. B. Taylor, J. Kim, J. Miller, et al., "The RAW microprocessor: a computational fabric for software circuits and general-purpose programs," *IEEE Micro*, vol. 22, no. 2, pp. 25–35, 2002.

[7] T. Marshall, L. Stansfield, J. Vuillemin, and B. Hutchings, "A reconfigurable arithmetic array for multimedia applications," in *Proceedings of the ACM/SIGDA 7th International Symposium on Field Programmable Gate Arrays*, pp. 135–143, 1999.

[8] Xilinx, http://www.xilinx.com/products/virtex4/index.htm.

[9] Xilinx, http://www.xilinx.com/products/virtex5/index.htm.

[10] Altera, http://www.altera.com/products/devices/stratix2/st2-index.jsp.

[11] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.

[12] R. Hartenstein, "A decade of reconfigurable computing: a visionary retrospective," in *Proceedings of IEEE Conference and Exhibition on Design, Automation and Test in Europe*, pp. 642–649, 2001.

[13] J. Becker and M. Vorbach, "Architecture, memory and interface technology integration of an industrial/academic configurable system-on-chip (CSoC)," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pp. 107–112, 2003.

[14] A. DeHon, Y. Markovsky, E. Caspi, et al., "Stream computations organized for reconfigurable execution," *Microprocessors and Microsystems*, vol. 30, no. 6, pp. 334–354, 2006.

[15] Y. Kim, M. Kiemb, C. Park, J. Jung, and K. Choi, "Resource sharing and pipelining in coarse-grained reconfigurable architecture for domain-specific optimization," in *Proceedings of Design, Automation and Test in Europe (DATE '05)*, vol. 1, pp. 12–17, 2005.

[16] J. Zawodny and P. Kogge, "Cache-in-memory," in *Innovative Architecture for Future Generation High-Performance Processors and Systems*, pp. 3–11, Maui, Hawaii, USA, January 2001.

[17] J. Draper, J. Sondeen, S. Mediratta, and I. Kim, "Implementation of a 32-bit RISC processor for the data-intensive architecture processing-in-memory chip," in *Proceedings of the IEEE Low Power Electronics and Design*, pp. 161–166, 2005.

[18] M. Lanuzza, M. Margala, and P. Corsonello, "Cost-effective low-power processor-in-memory-based reconfigurable datapath for multimedia applications," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 161–166, 2005.

[19] S. Khawam, T. Arslan, and F. Westall, "Synthesizable reconfigurable array targeting distributed arithmetic for system-on-chip applications," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, pp. 2051–2058, 2004.

[20] F. Bouwens, M. Berekovic, A. Kanstein, and G. Gaydadjiev, "Architecture exploration of the ADRES coarse-grained reconfigurable array," in *Springer Reconfigurable Computing: Architectures, Tools and Applications*, pp. 1–13, 2007.

[21] L. T. Smit, G. K. Rauwerda, A. Molderink, P. T. Wolkotte, and G. J. M. Smit, "Implementation of a 2-D 8×8 IDCT on the reconfigurable Montium core," in *Proceedings of International Conference on Field Programmable Logic and Applications (FPL '07)*, pp. 562–566, 2007.

[22] J. Balfour and W. J. Dally, "Design tradeoffs for tiled CMP on-chip networks," in *Proceedings of the 20th International Conference on Supercomputing*, pp. 187–198, 2006.

[23] L. Cannon, *A cellular computer to implement the kalman filter algorithm*, Ph.D. thesis, Montana State University, Bozeman, Mont, USA, 1969.

[24] V. Strassen, "Gaussian elimination is not optimal," *Numerische Mathematik*, vol. 13, pp. 354–356, 1969.

[25] S. M. Kang and Y. Leblebici, *CMOS Digital Integrated Circuits Analysis and Design*, McGraw-Hill, New York, NY, USA, 3rd edition, 2002.

[26] P. M. Heysters, G. J. M. Smit, and E. Molenkamp, "Energy-efficiency of the Montium reconfigurable tile processor," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA '04)*, pp. 38–44, 2004.

[27] D. Cronquist, C. fisher, M. Figueroa, P. Franklin, and C. Ebeling, "Architecture design of reconfigurable pipelined datapaths," in *Proceedings of the 20th Anniversary Conference on Advanced Research in VLSI*, pp. 23–40, 1999.