PAPER

Mapping Parallel FFT Algorithm onto SmartCell Coarse-Grained Reconfigurable Architecture

Cao LIANG[†], *Nonmember and* **Xinming HUANG**[†a)], *Member*

SUMMARY Fast Fourier Transform (FFT) is an important algorithm in many digital signal processing applications, and it often requires parallel implementation for high throughput. In this paper, we first present the SmartCell coarse-grained reconfigurable architecture targeted for stream processing. A SmartCell prototype integrates 64 processing elements, configurable interconnections, and dedicated instruction and data memories into a single chip, which is able to provide high performance parallel processing while maintaining post-fabrication flexibility. Subsequently, we present a parallel FFT architecture targeted for multi-core platforms computing systems. This algorithm provides an optimized data flow pattern that reduces both communication and configuration overheads. The proposed parallel FFT algorithm is then mapped onto the SmartCell prototype device. Results show that the parallel FFT implementation on SmartCell is about 14.9 and 2.7 times faster than network-on-chip (NoC) and MorphoSys implementations, respectively. SmartCell also achieves the energy efficiency gains of 2.1 and 28.9 when compared with FPGA and DSP implementations.

key words: *coarse-grained reconfigurable architecture, parallel FFT, energy efficiency, ASIC, FPGA, DSP*

1. Introduction

Nowadays, the real-time data streaming applications, especially over portable devices, often have stringent energy and performance requirements. The general solutions, such as Digital Signal Processors (DSPs), are widely used in conventional data-path oriented applications due to their flexibility and ease of use. However, they can not meet the increasing requirements on performance, cost and energy in data streaming applications due to their sequential software executions. On the other hand, application specific integrated circuits (ASICs) can provide best performance for specific applications, they generally have fixed data flow with predefined functionalities that makes them incapable of adapting to new system requirements or changes in standards. The long design cycle also becomes 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 DSPs and performance as of ASICs. The hardware-based RA implementation maps the computing tasks directly to the on-chip computing resources to avoid the software execution overheads, which results in both energy and performance gains over digital signal processors. Furthermore, RAs maintain the post fabric flexibility to reconfigure themselves for new system requirements or protocol updates, which are not feasible in ASIC implementations. Currently, field programmable gate arrays (FPGAs) are still the dominating technology in reconfigurable computing area. The most common SRAM-based FPGAs decompose complex logic functions into smaller ones and map them to Lookup Tables (LUTs) and other on-chip embedded resources. The island-style routing fabrics can be configured to form application-specific datapaths. 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, speed and configuration time, due to its huge routing area overhead and timing penalty. It is studied in [1] that an FPGA consumes about 14 times more dynamic power and is about 35 times larger than an equivalent ASIC when only logic elements are used.

In recognition of these issues, several research projects have been developed towards coarse-grained reconfigurable architectures (CGRAs) as summarized in [2]. They use coarse-grained (byte/word-wise) computing and communication components and target at data streaming application domain. Benefiting from much less routing overhead, these CGRA systems have potential advantages of better power efficiency compared with the fine-grained FPGAs while maintaining the flexibility. Among them, the RAW [3] system incorporates 16 simplified 32-bit MIPS processors in a 2D mesh structure to provide high parallel computing capacities. RaPiD [4] links a large number of heterogeneous reconfigurable components in a 1D array structure, including ALUs, multipliers, RAMs, etc. 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 [5] system, several reconfigurable pipeline stripes are offered as an accelerator for data-path processing. Limited configurable interconnection fabrics are developed, including a local inter-stripe network, unidirectional nearest neighbor connection between stripes and some global buses. The Matrix [6] 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. MorphoSys [7] is an integrated and configurable system-on-chip, targeted at high throughput and data parallel applications. A modified RISC processor is embedded to control the reconfigurable accelerating arrays with an

Manuscript received March 15, 2009.
Manuscript revised September 19, 2009.

[†]The authors are with the Department of Electrical and Computer Engineering, Worcester Polytechnic Institute, USA.
a) E-mail: xhuang@ece.wpi.edu

DOI: 10.1587/transele.E93.C.407

efficient memory interface between them. Some other reconfigurable architectures have also been implemented with various technologies, such as [8]–[10]. Most of them have been focused on exploration of computational models or efficient designs with respect to area and performance. Most recently, CGRA systems have also been extended to provide ultra low power consumption with limited computing resources, such as ADRES [11] and Montium [12].

SmartCell integrates some of the prominent features from previous systems. The 16-bit granularity is efficient for data parallelism, while keeping a low cost of computing and communication. SmartCell can be dynamic reconfigured to operate in SIMD, MIMD and systolic array styles. The hierarchical on-chip interconnection is another key feature that distinguishes SmartCell from other projects. The uniformed interconnection delay also eases the scheduling of the stream processing tasks among multiple cell units. In this paper, a novel parallel FFT algorithm is proposed and mapped onto the SmartCell system. Furthermore, the performance is evaluated and compared with other reconfigurable systems, including multi-processor architecture, MorphoSys, FPGA and DSPs.

The rest of this paper is organized as follows: Sect. 2 briefly presents the design of SmartCell, including architecture overview, interconnection design and configuration schemes. The design and mapping of a novel parallel FFT algorithm are presented in Sect. 3. Section 4 discusses the synthesis results and performance evaluations for the parallel FFT on SmartCell prototype device, followed by the conclusions in Sect. 5.

2. SmartCell Reconfigurable Architecture

SmartCell is developed as a coarse-grained reconfigurable architecture, targeted at domain specific applications with inherent data-parallelism, high computing and communication regularities. By integrating a large number of computing components and programmable switching fabrics onto the same chip, SmartCell has the potential to achieve the performance as of fixed function ASICs, meanwhile offering similar flexibility as of processors. More design details of SmartCell can be found in [13]. Currently, a new generation of SmartCell is developed with dedicated on-chip data memory for local data storage. In this section, we briefly discuss about the architecture design, on-chip interconnections and configuration schemes of SmartCell.

2.1 SmartCell Architecture Overview

An overview of SmartCell is depicted in Fig. 1. In a typical SmartCell system, a set of cell units are organized in 2D mesh structure. Each cell consists of four processing elements (PEs) placed in east, west, south and north directions to form a quad structure. Each PE consists of an arithmetic unit, a logic operator, I/O muxes and an instruction controller, as shown in Fig. 2(a). It can be configured to perform basic logic, shift and arithmetic functions with 16-bit granularity. Multiple PEs can be chained together for more complex functionalities. In current SmartCell design, a 1 Kb distributed data memory is attached to each PE. The PE and local memory connections are shown in Fig. 2(b). Each PE has full control of writing and reading of its own data memory. The data read from a memory can be shared between two PEs to provide more flexibility. Beside data memory, distributed instruction memory is used for each PE. It contains a group of 98-bit instruction codes that can be further segmented into computing, communication and instruction, and data memory controls. At runtime, an instruction code is loaded into the instruction controller on a cycle by cycle basis to build the dataflow for a specific algorithm. Unlike pipelined software execution in processors, a reconfigurable system utilizes the instruction code as configware to dynam-

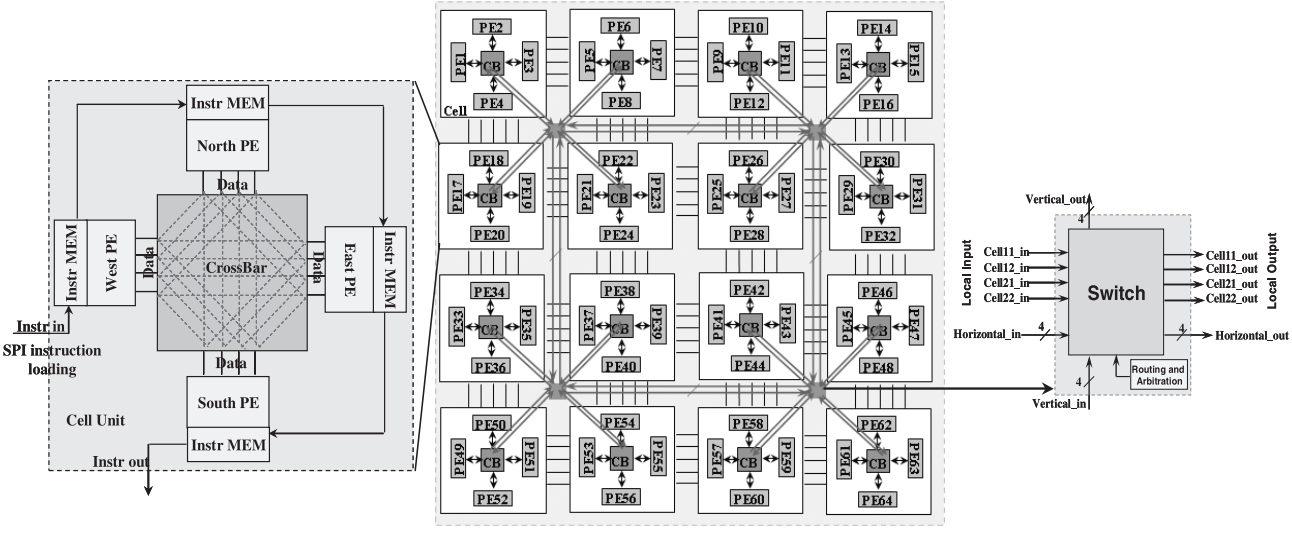

Fig. 1 Overview of the SmartCell architecture with integrated processing elements and hierarchical communications.

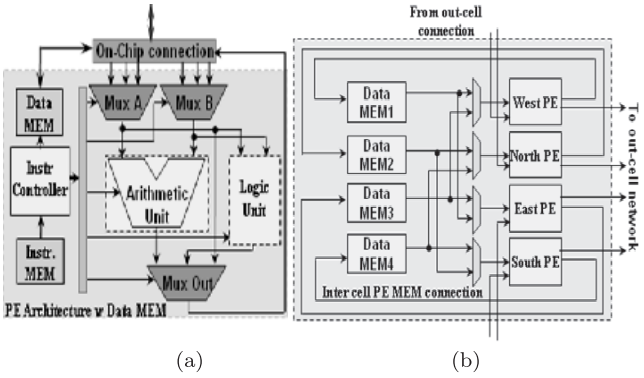

Fig. 2 PE structure and local data memory connection. (a) PE structure. (b) Inter cell PE and data memory connection.

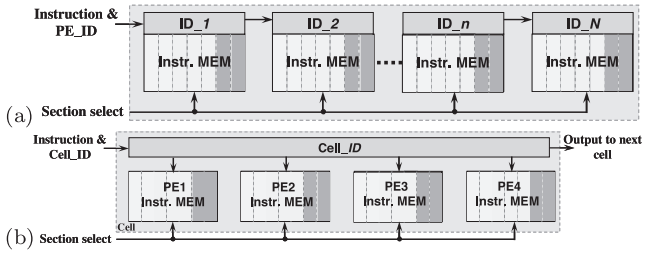

Fig. 3 Diagram of two modes of reconfigurations. (a) ID-based fine-grain configuration. (b) Cell broadcasting coarse-grain configuration. A global select signal is developed for memory partitioning.

ically restructure the hardware resources, which in turn reduces both computing and communication overheads.

2.2 Hierarchical On-Chip Interconnection

As the CMOS technology scales down, interconnection has become an increasingly important issue for integrated circuit design. The design of efficient data exchange scheme has become a key feature for high performance systems. In SmartCell, a three-level hierarchical routing structure is designed. At first, four PEs in the same cell are connected with a fully connected crossbar, which provides non-blocking arbitrary connections within the cell. Secondly, since cell units are tiled in a 2D mesh structure, the adjacent cells can be directly connected with short wires. The four PEs, placed at four edges in a cell, are directly linked to the nearest PE located in adjacent cells to form a static nearest neighbor network. It provides bi-directional data transmission with no extra delay and synchronization requirements. Finally, a hierarchical concentrated mesh (CMesh) network is developed to exchange data between nonadjacent cells, depicted in blue lines in Fig. 1. It is studied in [14] that the CMesh has the potential to provide best performance in terms of average latency and network efficiency among other NoCs, including Mesh, Torus, and FTree. The CMesh segments the network into clusters, with 4 cell units sharing the same switching component. In our design, each cell can receive a 36-bit signal through the CMesh network every clock cycle, which leads to a single hop communication rate of 57.6 Gb/s for a 4 by 4 SmartCell operating at 100 MHz.

2.3 SmartCell Configuration Process

A serial peripheral interface (SPI) is designed to configure and update the instruction memories, as shown in top right of Fig. 1. The instruction memories are linked in a ripple array style with the inputs and outputs chained one to another. During the initial configuration procedure, the instruction code is loaded into the first PE's instruction memory and then shifted down to the second one and so on. This procedure stops after the last active PE is configured. The run time reconfiguration is achieved by the same SPI structure. Two modes are provided for fine-grained ID-based configuration and coarse-grained cell broadcasting, as shown in Fig. 3(a) and (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 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. It's noteworthy that the propagation latency is not the same for all PE/cell units along the SPI chain: the closer to the input port, the faster the configuration can be done. To compensate for 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 ones. Once the new contexts are fully loaded, a global select signal is used to indicate the switch of operations. The configuration latency is hidden by this means, which allows multiple operations to be efficiently swapped in one clock cycle.

3. Parallel FFT Algorithm and Mapping onto SmartCell

Discrete Fourier Transform (DFT) is one of the most important digital signal processing algorithms in many communication and multimedia applications. An N-point DFT is defined in Eq. (1).

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \tag{1}$$

where $x(n)$ and $X(k)$ are the complex input and output, and the twiddle factor $W_N = e^{-2\pi i/N}$. Fast Fourier Transform (FFT) is a fast DFT algorithm that reduces the computing complexity from $O(N^2)$ to $O(Nlog_2(N))$. However, the intensive computations are still the bottleneck for large-size FFT/IFFT designs. Recently, several approaches have been proposed to compute the FFT in parallel on multi-processor

architecture [15]–[19]. In this section, we propose a novel parallel FFT algorithm with evenly distributed computation tasks and optimized data transfer pattern among the processing units. It can be efficiently mapped onto SmartCell and easily configured for different size of FFT transforms.

3.1 Two-Step Parallel FFT Algorithm

The Decimation-in-Time (DIT) Radix-2 Cooley-Tukey FFT [20] is chosen to be mapped onto the SmartCell architecture. Radix-2 FFT partitions the calculation of N-point DFT into $log_2(N)$ stages, each of which consists of a group of $N/2$ 2-point DFTs, also called butterfly units (BUs). The complex number butterfly calculation is the basic operation involved in Radix-2 FFT. Comparing with the Decimation-in-Frequency (DIF) FFT, the DIT algorithm achieves a more balanced computing load between the two branches in the butterfly operation. It also allows the input data to be decomposed into smaller sequences and to be processed independently, which makes it more popular in parallel FFT designs [15], [21]. Although Radix-4 and Radix-8 FFTs require fewer computing stages, they are not adopted in our design due to higher configuration and communication complexities.

Two steps are developed in the proposed algorithm to compute the FFT for a data block: local sequential execution and cross parallel execution. Before the transform begins, the N points input data are evenly distributed to the P butterfly processing units, with N/P points stored in each BU. During the local sequential execution, the first $log_2(N/2P)$ stages are computed sequentially in each BU operating on N/P locally stored data. Since each butterfly operation consumes 2 points at a time, we assume both N and P are powers of 2 and satisfy $N \geq 2P$ and $P > 1$, without loss of generality. After the sequential step, a cross parallel execution is performed for the remaining $log_2(2P)$ stages of N-point FFT. Cross communication among different BUs are necessary to exchange data that their indices are more than N/P apart.

$$f(c_1, c_2, w) = c_1 + w \times c_2 \tag{2}$$

Algorithm 1 : Local sequential $log_2(N/2P)$ Stages

```
1: Input: c = c[0], c[1],..., c[N/P − 1]
2: Output: c = c[0], c[1],..., c[N/P − 1]
3: for i ← 0 to log2(N/2P) − 1 do
4:     for j ← 0 to N/2P − 1 do
5:         get w1, w2 based on i, j
6:         c[j] = f(c[2j], c[2j + 1], w1)
7:         c[j + N/2P] = f(c[2j], c[2j + 1], w2)
8:     end for
9: end for
```

Algorithm 1 demonstrates the operations involved in the local sequential execution for each BU. Before the transform starts, the N-point signals are bit reversed and partitioned into P sub-blocks. Each N/P sub-block is fed to one butterfly unit sequentially, denoted as $c[0]$ to $c[N/P - 1]$ in the algorithm. After the data are completely loaded, the first $log_2(N/2P)$ stages of the original N-point FFT is computed locally inside each BU. The butterfly function $f(c_1, c_2, w)$ is defined in Eq. (2), where c_1 and c_2 are two input branches, and w is the twiddle factor. Two input data are read out from two consecutive memory locations and are written back to two locations that are $N/2P$ apart after processing. Two sets of memory blocks have been used to swap data between stages to avoid data corruptions. All calculations are completed using local data only. Thus no cross BU communications occur during the sequential step. Data level parallelism is achieved by computing P sub-blocks concurrently in different BUs.

Algorithm 2 : Cross Parallel $log_2(2P)$ Stages

```
1: Input: c = c[0], c[1],..., c[N/P − 1]
2: Output: c = c[0], c[1],..., c[N/P − 1]
3: for i ← 0 to log2(2P) − 1 do
4:     for j ← 0 to N/2P − 1 do
5:         if (N > 2P) then
6:             if (i == 0) then
7:                 t1 = c[2j]
8:                 t2 = c[2j + 1]
9:             else
10:                t1 = c[j]
11:                t2 = c[j + N/2P]
12:            end if
13:            get w1, w2 based on i, j and BU_ID
14:            k = BU_ID
15:            r1 = f(t1, t2, w1)
16:            r2 = f(t1, t2, w2)
17:            if (k mod 2 == 0) then
18:                send r1 → c[j] of (k/2)^th BU
19:                send r2 → c[j] of (k/2 + P/2)^th BU
20:            else
21:                send r1 → c[j + N/2P] of ((k − 1)/2)^th BU
22:                send r2 → c[j + N/2P] of ((k − 1)/2 + P/2)^th BU
23:            end if
24:        else
25:            get w1, w2 from i and BU_ID
26:            k = BU_ID
27:            r1 = f(c[0], c[1], w1)
28:            r2 = f(c[0], c[1], w2)
29:            if (k mod 2 == 0) then
30:                send r1 → c[0] of (k/2)^th BU
31:                send r2 → c[0] of (k/2 + P/2)^th BU
32:            else
33:                send r1 → c[1] of ((k − 1)/2)^th BU
34:                send r2 → c[1] of ((k − 1)/2 + P/2)^th BU
35:            end if
36:        end if
37:    end for
38: end for
```

The cross parallel step performs the remaining $log_2(2P)$ stages of the original N-point FFT, as shown in Algorithm 2. During this step, the intermediate butterfly results need to be transferred among BUs. We use BU_ID to represent the index number of current BU ranging from 0 to $P - 1$. Two cases are considered based on the number of FFT points N and the number of butterfly units P. When N

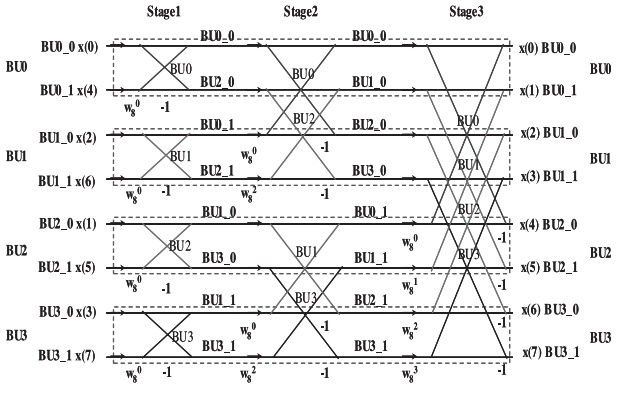

Fig. 4 An example of data transfer pattern for 8-point FFT.

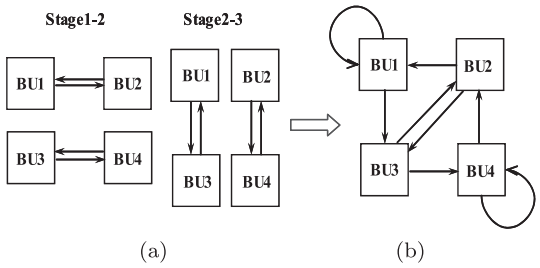

Fig. 5 Data transfer pattern for 8-point FFT. (a) Traditional communication pattern. (b) Optimized fixed data flow.

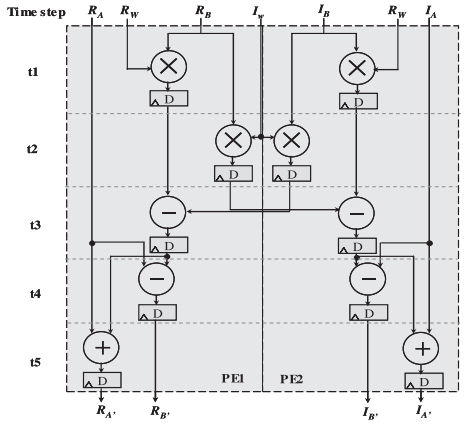

Fig. 6 Mapping of butterfly operation onto two PEs of SmartCell.

is equal to $2P$, each BU only processes two points involved in the butterfly operation. No sequential step is necessary in this case. The computing and data transfer pattern are listed in Line 25 to 35 in Algorithm 2. When N is greater than $2P$, more than 2 points are stored and processed by each BU, which requires both sequential and parallel steps. The related cross parallel execution is listed in Line 6 to 23. The initial iteration($i == 0$) calculates the last stage of the local N/P-point FFT. After that, the data fetching pattern is changed to read from two memory locations that are $N/2P$ apart due to cross BU communications. Generally, in the cross parallel execution, each processor calculates the butterfly results from its local data and then forwards the results to two processors based on its own *BU_ID*.

3.2 Data Transfer Pattern For Parallel FFT

Besides the computing tasks, the data transfer pattern is the other important consideration for high performance parallel FFT design. Efficient data transfer plays a key role in reducing communication delay and overhead, which in turn improves the overall system throughput. In this subsection, we analyze the data transfer pattern during the cross parallel execution.

The data transfer is described in Line 17 to 23 and Line 29 to 35 in Algorithm 2. Given the total number of points N and processors P, the destination BUs are only determined by the current processor ID k, while the destination memory locations are determined by the ID k and the iteration number j. The destination BUs and memory locations are independent of the current FFT stage i. Thus the proposed FFT algorithm achieves a fixed data transfer pattern at all FFT stages. Figure 4 gives a data flow example of 8-point FFT with 4 butterfly units. Each BU stores two input data in bit reversed order. According to the algorithm, BU0 calculates the butterfly results from its local data and sends them to memory 0 of BU0 and BU2. The same data flow is repeated during stage 2 and 3. Other BUs follow similar fixed data transfer pattern as depicted in the figure. The two-line crossings denoted with the same label represent the butterfly operations executed by the same BU during different stages. The traditional data transfer pattern for 8-point Radix-2 FFT is drawn in Fig. 5(a). In this case, the computing tasks are horizontally partitioned among the BUs, as highlighted in dash-line boxes in Fig. 4. From stage 1 to stage 2, data are transferred between (BU0, BU1) and (BU2, BU3) pairs. The pattern changes to (BU0, BU2) and (BU1, BU3) pairs during the transition from stage 2 to stage 3. In contrast, a fixed data transfer pattern is achieved for all transform stages in the proposed parallel FFT algorithm, as shown in Fig. 5(b). The optimized transfer pattern reduces both communication and configuration overheads. Experimental results show that, for a 4 by 4 SmartCell system, the fixed data transfer algorithm reduces $2/(log_2(N) + 2)$ of total communication overhead with processing point for $N \geq 64$.

3.3 Mapping FFT onto SmartCell

The Radix-2 FFT butterfly calculation is formulated in Eq. 3. The complex number operation can be optimized into four real multiplications and six real additions. In our design, two PEs are used to perform the butterfly operation in a sequential manner. Figure 6 depicts the implementation with data flow scheduling. The real and imaginary parts of the output signals can be processed in parallel. The intermediate results are shared between these two PEs through the inter cell crossbar unit. In this way, two butterfly units can be mapped onto the 4 PEs in the same cell. To perform FFT on SmartCell, the input data are partitioned into consecutive chunks, each of which is then loaded into the data memories

of two PEs performing the same butterfly operation. The sequential execution step starts after the input data are fully loaded. Since the data processing is isolated inside the local PEs, no cross cell communications are involved during this step. After the local sequential step is finished, the FFT butterfly needs to transfer data between PEs in multiple cells. Due to the fixed communication pattern, the intermediate results can be efficiently exchanged through either nearest neighbor connection or CMesh network within 1 hop range for any point FFTs in our experimental 4 by 4 SmartCell system. No extra communication and configuration overheads are involved.

$$\begin{aligned} R_{A'} &= R_A + R_B R_W - I_B I_W \\ I_{A'} &= I_A + I_B R_W + R_B I_W \\ R_{B'} &= R_A - R_B R_W + I_B I_W \\ I_{B'} &= I_A - I_B R_W - R_B I_W \end{aligned} \tag{3}$$

Scalability is an important performance criterium in reconfigurable system designs. For the proposed parallel FFT scheme, SmartCell is able to implement FFT of any sizes as long as the input data can be fully stored in the on-chip data memories. Larger size FFTs are able to be implemented if using an external memory. By changing several loop control parameters, SmartCell can be reconfigured to compute a different size of FFT. To analyze its performance, a 1024-pt FFT is mapped onto a 4 by 4 SmartCell with 1 K data memory attached to each PE. In this case, 32 butterfly units are available to compute the FFT for parallel processing. The first 4 stages of the 1024 FFT are carried out locally in each cell. 6 cycles are needed for each butterfly operation, with 5-cycle computing and 1-cycle storing data. During the cross parallel execution, the butterfly operation still takes 6 cycles to finish since all communication is within one hop in CMesh network. Simulation results show that a block of 1024-pt FFT can be finished in 992 clock cycles, which include 32 cycles for initial data input. This leads to a system throughput of 101 KBlocks/s operating at 100 MHz.

4. SmartCell Synthesis Results and Benchmark Comparisons

A 4 by 4 prototype SmartCell with 64 PEs is designed and synthesized in standard cell ASIC design flow. The area and timing performance is provided based on the synthesis reports. The proposed parallel FFT algorithm is manually mapped to the prototype system to evaluate its performance. Up to 1024-pt FFT can be directly implemented in our current design. We then evaluate the system throughput and energy efficiency performance among SmartCell, FPGA and DSPs. The SmartCell system throughput is further compared with other parallel FFT implementations, including NoC [15] and MorphoSys [18]. The reason to choose NoC and MorphoSys is that they implement parallel FFTs onto the same number of processor elements as in SmartCell system.

Table 1 System design environment and simulation setups.

System dimension	4 by 4 with 64 K data memory
Design tools	ModelSim, Synopsys CAD tools
Library	TSMC 90*nm* process
Synthesis Environment	Worst case condition
Voltage	1 V
Simulation frequency	100 MHz

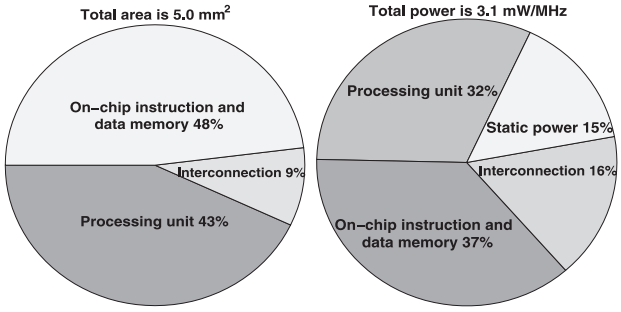

Fig. 7 Area and power breakdown of the prototype SmartCell.

4.1 SmartCell Synthesis Results

The prototype SmartCell system is developed and synthesized with standard CAD tools. A functional RTL model is firstly designed in hardware description language (HDL) and then synthesized in Synopsys DesignCompiler to generate the CMOS standard cell ASIC using TSMC 90*nm* technology. The area and timing results are generated by DesignCompiler using worst case conditions. Some experimental setups are listed in Table 1.

According to the synthesis results, the prototype SmartCell occupies about 5.0 mm^2, which is about 2.0 million gates. The system area, breakdown into PE, on-chip memory and interconnections, are shown in Fig. 7. About half of the total area is consumed by the on-chip data and instruction memories. The processing units and the hierarchical interconnection networks roughly consist of 43% and 9% in total area, respectively. A worst case maximum frequency of 202 MHz can be achieved in the prototype system. Again, customized components can be used to improve the timing results, such as pipelined multiplier and carry lookahead adder. At runtime, register delay can be inserted between MULT and ADD components to break the MAC operation into 2 cycles. Experiments show that the proposed FFT is able to run at up to 295 MHz. In terms of dynamic reconfiguration, SmartCell can be reconfigured to perform a different size FFT in 90 clock cycles, which is within 1μs at 100 MHz.

The power consumption of SmartCell for the evaluated FFT benchmarks is estimated in Synopsys PowerCompiler based on netlist annotation from gate level simulations. Clock gating is automatically added by synthesis tool to dynamically turn off the clocks for unused registers. Figure 7 shows the processing units consume about 32% of total power, while the on-chip memories consume about 37%

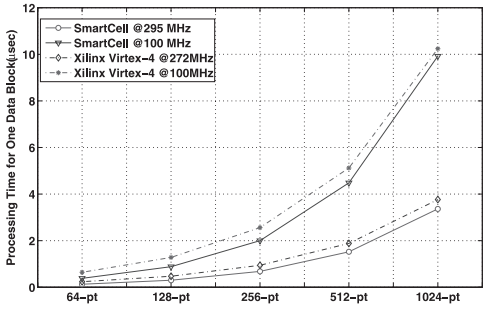

Fig. 8 Processing time comparison between SmartCell and FPGA.

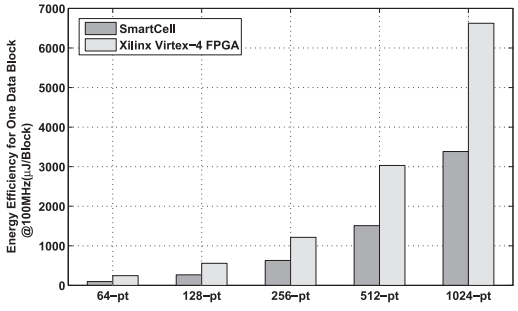

Fig. 9 Energy consumption comparison between SmartCell and FPGA.

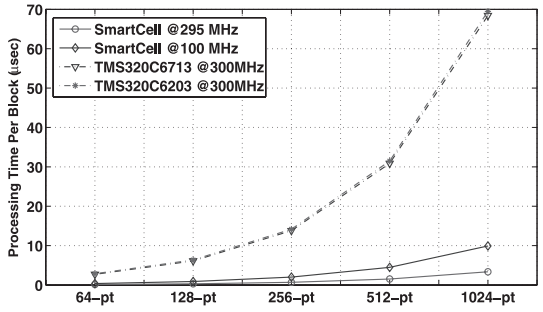

Fig. 10 Processing time comparison between SmartCell and TI DSPs.

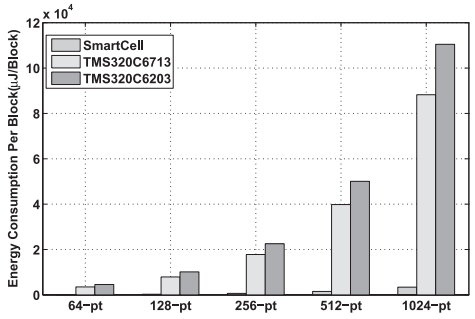

Fig. 11 Energy consumption comparison between SmartCell and TI DSPs.

of the total power. Due to technology shrink, the static power increased from 6% to 15% comparing with our first implementation in [13]. On average, SmartCell consumes 3.1 mW/MHz for the evaluated FFT benchmarks.

4.2 Comparisons with FPGA Implementations

In this section, we compare the system throughput and energy efficiency results between SmartCell and FPGAs. The Xilinx's 90 *nm* Virtex-4 XC4VLX15 FPGA [22] is selected as the benchmark platform, which is the smallest FPGA in Virtex-4 series. The FFT of different sizes is generated from CoreGen [23] in Xilinx's ISE 10.1 CAD tool. Pipelined FFT architecture is adopted in the FPGA implementations with $log_2(N)$ butterfly stages chained in a pipeline form. Synthesis results show that a maximum frequency of 272 MHz can be achieved by FPGA for the evaluated FFT benchmarks.

Figure 8 compares the per-block FFT processing time between FFT and SmartCell. SmartCell is able to compute a 64-pt FFT in 38 cycles, while 64 cycles are needed in FPGAs. With both running at the maximum frequency, SmartCell achieves a maximum throughput gain of a factor 1.8 compared with FPGA. On average, SmartCell is about 1.3 times faster than the FPGA implementations for the evaluated benchmarks. Furthermore, due to its unbalanced memory loads and intensive control requirements, the pipelined FFT is not suitable for reconfigurable SmartCell architecture.

Figure 9 compares the FFT energy consumption for one data block between SmartCell and FPGA operating at 100 MHz. The SmartCell power consumption is estimated in Synopsys PowerCompiler based on the netlist annotations. The XPower Analyzer is used to evaluate the FPGA power consumption also based on the switching annotation from gate level simulations. Only the core power consumption is recorded in FPGA designs for fair comparison. In 64-pt FFT, SmartCell consumes 63.1% less energy than FPGA, since no data memory are used in SmartCell for this case. On average, SmartCell is about 2.1 times more energy efficient than benchmarked fine-grained FPGA. Results show that SmartCell achieves an energy efficiency of about 20.6 GOPS/W for all benchmarks under test.

4.3 Comparisons with DSP Implementations

The system throughput and energy efficiency for the FFT benchmarks are also compared between SmartCell and DSPs, including TI's TMS320C6203™ and TMS320C6713™ [24]. The reason to choose these DSPs to compare is that they are targeted at high performance signal processing applications and the Radix-2 FFT benchmark results are provided by the vendor. Figure 10 compares the FFT processing time for one data block between SmartCell and DSPs. We assume the DSPs are operating at the highest frequency specified in data sheet. In terms of cycle counts, TMS320C6713 needs 20522 cycles to compute 1024-pt FFT, while SmartCell only requires 992 cycles. When the maximum frequency is used, SmartCell is about 20.8 times faster than the DSP-based implementations. According to data sheet, the typical core power consumption for TMS320C6203 and TMS320C6713 is about 5.3 mW/MHz and 4.3 mW/MHz. Figure 11 compares the per-block energy consumption between SmartCell and DSPs. On aver-

Table 2 Cycle counts comparison among different parallel FFT platforms.

FFT Size	SmartCell	Morphosys [18]	NoC [15]
64	38	111	-
128	88	225	-
256	200	520	4827
512	448	1222	5702
1024	992	2613	7726

age, SmartCell is about 36.8 and 28.9 times more energy efficient than TMS320C6203 and TMS320C6713 DSPs, respectively.

4.4 Cycle Count Comparison with Other Parallel FFT Implementations

At last, we present the cycle count comparison with other parallel FFT implementations, including NoC [15] and MorphoSys [18] implementations. The same number of 64 PEs is integrated in both MorphoSys and NoC systems with 2D mesh interconnections. Due to different operating frequencies, the number of clock cycles used to complete the FFT computations is compared. Table 2 lists the system throughput among different platforms. SmartCell outperforms both NoC and MorphoSys platforms in all benchmark FFTs regarding on system throughput. On average, SmartCell is about 14.9 times faster than the NoC FFT implementations. Similarly, SmartCell provides about 2.7x throughput gain if compared with the coarse-grained MorphoSys implementation. This is primarily due to reduced communication overhead by the proposed parallel FFT algorithm.

5. Conclusions

In this paper, a novel two-step parallel FFT algorithm is proposed to provide a balanced workload and fixed data flow pattern across all processing units for multi-processor platforms. It is then mapped onto the SmartCell coarse-grained reconfigurable architecture with distributed data memories. A prototype SmartCell system with 64 PEs is implemented in standard cell ASICs with TSMC 90 *nm* technology. The proposed FFT algorithm with different sizes between 64 to 1024 points are mapped onto the prototype SmartCell device. For the evaluated FFT benchmarks, SmartCell dissipates about 310 mW of power consumption operating at 100 MHz. Compared with other FFT implementations, SmartCell architecture is about 14.9 and 2.7 times faster than the parallel FFT results in NoC and MorphoSys, respectively. SmartCell is also about 2.1 and 28.9 times more energy efficient than the FPGA and DSP based implementations. We claim that SmartCell is a promising reconfigurable and energy efficient platform for stream processing.

Acknowledgements

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] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," IEEE Trans. Comput.-Aided Des. Integer. Circuits Syst., vol.26, no.2, pp.203–215, 2007.
[2] R. Hartenstein, "A decade of reconfigurable computing: A visionary retrospective," Proc. IEEE Conference and Exhibition on Design, Automation and Test in Europe, pp.642–649, 2001.
[3] M. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The RAW microprocessor: A computational fabric for software circuits and general-purpose programs," IEEE Micro, vol.22, no.2, pp.25–35, 2002.
[4] C. Fisher, K. Rennie, G. Xing, S. Berg, K. Bolding, J. Naegle, D. Parshall, D. Portnov, A. Sulejmanpasic, and C. Ebeling, "Emulator for exploring RaPiD configurable computing architectures," Proc. 11th International Conference on Field-Programmable Logic and Applications, pp.17–26, 2001.
[5] H. Schmit, D. Whelihan, A. Tsai, M. Moe, B. Levine, and R. Taylor, "PipeRench: A virtualized programmable datapath in 0.18 micron technology," Proc. IEEE Custom Integrated Circuits Conference, pp.63–66, 2002.
[6] E. Mirsky and A. DeHon, "MATRIX: A reconfigurable computing architecture with configurable instruction distribution and deployable resources," Proc. IEEE Symposium on FPGAs for Custom Computing Machines, pp.157–166, 1996.
[7] H. Singh, M. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, and E.C. Filho, "MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications," IEEE Trans. Comput., vol.49, no.5, pp.465–481, 2000.
[8] J. Becker and M. Vorbach, "Architecture, memory and interface technology integration of an industrial/academic configurable system-on-chip (CSoC)," Proc. IEEE Computer Society Annual Symposium on VLSI, pp.107–112, 2003.
[9] A. DeHon, Y. Markovsky, E. Caspi, M. Chu, R. Huang, S. Perissakis, L. Pozzi, J. Yeh, and J. Wawrzynek, "Stream computations organized for reconfigurable execution," Elsevier Microprocessors and Microsystems, vol.30, no.6, pp.334–354, 2006.
[10] Y. Kim, M. Kiemb, C. Park, J. Jung, and K. Choi, "Resource sharing and pipelining in coarse-grained reconfigurable architecture for domain-specific optimization," Proc. IEEE Design, Automation and Test in Europe, pp.12–17, 2005.
[11] F. Bouwens, M. Berekovic, A. Kanstein, and G. Gaydadjiev, "Architecture exploration of the ADRES coarse-grained reconfigurable array," Springer Reconfigurable Computing: Architectures, Tools and Applications, pp.1–13, 2007.
[12] L. Smit, G. Rauwerda, A. Molderink, P. Wolkotte, and G. Smit, "Implementation of a 2-D 8x8 IDCT on the reconfigurable Montium core," Proc. 2007 International Conference on Field Programmable Logic and Applications (FPL'07), pp.562–566, 2007.
[13] C. Liang and X. Huang, "SmartCell: A power-efficient reconfigurable architecture for data streaming applications," Proc. IEEE Workshop on Signal Processing Systems (SiPS'08), pp.257–262, 2008.
[14] J. Balfour and W. Dally, "Design tradeoffs for tiled CMP on-chip networks," Proc. 20th annual international conference on Supercomputing, pp.187–198, 2006.
[15] J. Bahn, J. Yang, and N. Bagherzadeh, "Parallel FFT algorithms on network-on-chips," Proc. IEEE Fifth International Conference on Information Technology, pp.1087–1093, 2008.
[16] Q. Zhang, A. Kokkeler, and G. Smit, "An efficient FFT for OFDM

based cognitive radio on a reconfigurable architecture," Proc. IEEE International Conference on Communications, pp.6522–6526, 2007.
[17] D. Kim, M. Kim, and G. Sobelman, "Parallel FFT computation with a CDMA-based network-on-chip," Proc. IEEE International Symposium on Circuits and Systems, pp.1138–1141, 2005.
[18] A. Kamalizad, C. Pan, and N. Bagherzadeh, "Fast parallel FFT on a reconfigurable computation platform," Proc. IEEE 15th Symposium on Computer Architecture and High Performance Computing, pp.254–259, 2003.
[19] Y. Lin and C. Lee, "Design of an FFT/IFFT processor for MIMO OFDM systems," Proc. IEEE Trans. Circuits and Systems I, pp.807–815, 2007.
[20] J. Cooley and J. Tukey, "An algorithm for the machine calculation of complex Fourier series," Math. Comput, pp.297–301, 1965.
[21] C. Zhong, G. Han, and M. Huang, "Some new parallel fast fourier transform algorithms," Proc. IEEE Parallel and Distributed Computing, Applications and Technologies, pp.624–628, 2005.
[22] xilinx.com/support/#Virtex 4
[23] xilinx.com/support/software/coregen/61i_coregen_examples.htm
[24] focus.ti.com/dsp/docs/dsphome.tsp?sectionId=46

Cao Liang received the M.S. degree in electrical engineering from University of New Orleans and the B.S. degree in Communication and Information Engineering from University of Electronic Science and Technology of China. He is currently working toward the Ph.D. degree in Electrical and Computer Engineering at Worcester Polytechnic Institute, MA. His research interests include high performance computing architecture design, VLSI design, reconfigurable systems, and routing in collaborative computing networks.

Xinming Huang is an Assistant Professor in the Department of Electrical and Computer Engineering at Worcester Polytechnic Institute (WPI), Worcester, MA. He received the Ph.D. degree in electrical engineering from Virginia Polytechnic Institute and State University, Blacksburg, VA in 2001. He was a member of technical staff with the wireless advanced technology laboratory, Bell Labs of Lucent Technologies, from 2001 to 2003. He was also an Assistant Professor with the Department of Electrical Engineering at the University of New Orleans from 2003 to 2006. He was among the recipients of the DARPA/MTO young faculty award in 2007, the IBM faculty fellowship award in 2004, and the central Bell Labs annual excellence and teamwork award in 2002. His research interests are in the areas of circuit design and system architectures for reconfigurable computing, wireless communications, and networked embedded systems.