# Generic Multi-Phase Software-Pipelined Partial-FFT on Instruction-Level-Parallel Architectures and SDR Baseband Applications

Min Li†, David Novo†, Bruno Bougard†, Liesbet Van Der Perre† Francky Catthoor†
† Nomadic Embedded System Division, IMEC, Leuven, Belgium
Email: {limin, novo, bougardb, vdperre, catthoor}@imec.be

***Abstract*—The PFFT (Partial FFT) is an extended FFT where only part of input or output bins are used. By pruning the useless dataflow, the PFFT can potentially achieve a significant speedup in many important applications. Although theoretical aspects of the PFFT have been thoroughly studied in past three decades, efficient implementations were rarely reported. The most important obstacle is the highly irregular dataflow and the associated control flow. In addition, a size-$N$ PFFT has $2^N$ dataflow possibilities, so that delivering both flexibility and efficiency in the same implementation is very challenging. This paper presents a generic scheme to map the highly irregular dataflow of arbitrary PFFT onto ILP architectures with highly efficient SWP (SoftWare-Pipelining). Constraints and opportunities of algorithms and architecture are carefully analyzed and exploited. We introduce a multi-phase partitioning, bringing heterogeneous control structures and heterogeneous software pipelining schemes to minimize control overheads and to maximize the efficiency of SWP. The proposal has been tested with 10 representative benchmarks extracted from baseband applications. In experiments cycle-counts, instructions, NOPs, L1D/L1P access/miss/hit are thoroughly analyzed. Comparing to full FFTs with efficient SWP, our work reduces 20.5% - 87.5% cycle-counts, 11.2% - 86.5% instructions, 16.1% - 79.4% L1D cache accesses and 19.5% - 87.1% L1P cache accesses. To the best of our knowledge, this is the first reported work about the generic software-pipelined PFFT on ILP architectures.**

## I. Introduction

The FFT (Fast Fourier Transform) is doubtlessly one of the most important and fundamental techniques in the signal processing world. Among the countless hardware and software implementations, the vast majority of them handle the cases in which the entire set of the input/output bins are required. However, in many real-life applications this is *not* the case. Long lists of examples can be found in the references of this paper [1-2][4-11]. These partial output/input cases are extraordinarily important for the future wireless systems [1], such as the OFDMA (Orthogonal Frequency Division Multiplexing Access) and cognitive radio [2].

Since we do not want to pay the price of the full FFT for only part of the input/output bins, the redundant dataflow can be pruned *without* changing the I/O behaviors. This is called *PFFT* (Partial FFT). A small example of PFFT dataflow graph is shown in Fig.1. In many applications the percentage of required input/output bins is very small. For instance, in the 3GPP LTE (Long Term Evolution), if the OFMDA symbol size is 1024 and 12 users equally share the available 600 sub-carriers, only 50 of the 1024 FFT output bins (4.88%) are required for each mobile terminal [3]. Hence, the computation/memory-access operations and shuffling operations saved by the PFFT on practical platforms will be very attractive.

Unfortunately, although the research of the PFFT can be traced back to 1971 [4] and the theoretical aspects of the PFFT have been thoroughly studied in the last three decades [1-2][4-11], there were not many breakthroughs in the implementation. We find very few papers about the PFFT in real-life systems. The most important obstacle is that the data-flow and control-flow in the PFFT destroy the full and nice regularity of the FFT, so that efficient implementations on parallel architectures become very difficult. The irregularity can be can be observed in Fig.1. In addition, a size-$N$ PFFT have $2^N$ possibilities regarding the input or output patterns. Clearly, delivering full-flexibility and high-efficiency at the same time is very difficult. These challenges are imposed on not only hardware implementations but also software-based implementations.

Our work targets parallel programable architectures. In deep sub-micron era, the none-recurring-engineering-cost soars up and the time-to-market stress drastically increases. Hence, leveraging programmable architectures and implementing signal processing in softwares are becoming popular in many important application areas, such as the audio processing, the video processing and the the SDR (Software Defined Radio) [12]. Especially, the VLIW (Very Long Instruction Word) [13] and the CGA (Coarse Grain Array) [14], as two of the ILP (Instruction Level Parallel) architecture instances, have clearly shown their promising potentials.

This paper presents a *generic* scheme to map the highly irregular dataflow of the PFFT onto ILP architectures with *efficient SWP* (SoftWare-Pipelining).

To eliminate the hardware and energy overhead of the run-time instruction scheduler for ILP, the SWP was proposed as an effective and efficient compilation/scheduling technique to optimize the ILP resource-utilization at design-time [15]. The key idea of the SWP is, if the loop is qualified, initializing loop iteration $i+1$ without completely finishing iteration $i$ with a fixed interval. In this way, multiple loop-iterations will overlap with each other, executing simultaneously on ILP architectures.

However, SWP of the PFFT is not straightforward at all.

978-3-9810801-3-1/DATE08 © 2008 EDAA

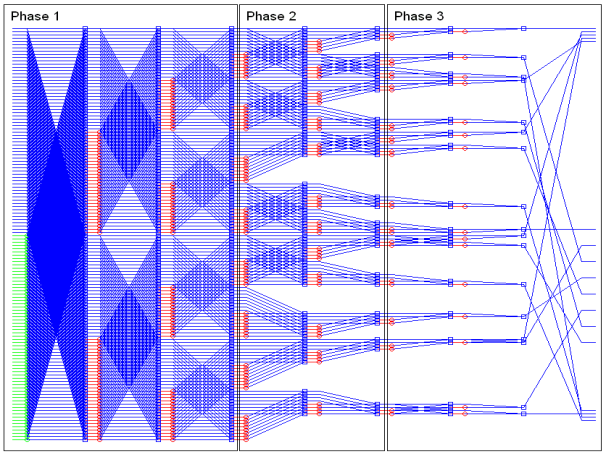



Fig. 1. Simple Example of Dataflow Graph in Partial FFT

The SWP is very sensitive to the regularity in loops [15]. Irregular dataflow in the innermost loop might disqualify the loop for the SWP. Unfortunately, in many important PFFT applications such as the multi-resolution spectrum-scanning and practical OFDMA demodulators, the PFFT data-flow is indeed highly irregular. Hence, when developing a generic scheme for software pipelining an arbitrary PFFT, the challenges are not trivial.

In our work, opportunities and constraints of algorithms and architectures are thoroughly analyzed and exploited. First, different FFT dataflow variants are exploited to enable aggressive optimizations in subsequent steps. Then, we propose to partition the PFFT into three phases, so that *heterogeneous* control flow structures and *heterogeneous* SWP schemes are elaborately combined inside the PFFT. The control structures work on a very compact table with encoded dataflow information, so that the flexibility is fully supported. Our proposal achieves significant reductions in terms of cycle-counts, instructions, L1D/L1P accesses/misses and so on. To the best of our knowledge, this is the first reported work about the generic software-pipelined PFFT on ILP architectures.

The remaining part of the papers consists of the following sections: Section II surveys the related work; Section III present our work; Section IV brings experiment results and detailed analysis; Section V concludes the paper.

## II. Related Work

The PFFT research can be traced back to 1971. In [4] the author first presented the idea of pruning useless dataflow in FFTs. Later on, the theoretical aspects of PFFT had been extensively studied in different application contexts [1-2][4-11]. In [9] the frequency-shift property is used to increase complexity-reduction. Various dataflow pruning algorithms are studied and compared in [8]. In [2] the complexity reduction is analyzed in the context of OFDM. In [1] the lower and upper bounds of complexity-reductions are analyzed. The above work focuses on the abstract complexity analysis, no implementation aspects are not taken into account.

In [10] a bit-serial systolic-array based ASIC (Application Specific Integrated Circuit) of PFFT is presented. The dataflow is pruned with the help of the LDIF (Lattice Decomposition in Frequency). The application context in [10] is OFDM systems with a single-block output. However, it is not clear whether the systolic array is flexible enough to handle arbitrary PFFT. In [[6]] the general FFT pruning algorithm was introduced with some implementation considerations. The proposed implementation scheme is based on a $N \times log_2 N$ table with $0/1$ elements, and a "1" means a valid/useful dataflow. A modified scheme is introduced in [11] for OFDM systems. A software implementation is also introduced in this paper, but the targeted platform is a desktop processor, not a low power ILP architecture.

Although the SWP of PFFT is not studied yet, the SWP of full FFTs is very mature. The SWP of a full FFT is based on the hierarchical structure. A Radix-2 Cooley-Tukey full FFT is hierarchically composed by the following elements: (1) *Butterfly*: a pair of operation defined by a twiddle factor on a given pair of memory locations. (2) *Butterfly-groups*: a group of butterflies that are next to each other in a consecutive way. (3)*Stages*: a Radix-2 Cooley-Turk FFT consists of $log_2 N$ stages. Each stage of the FFT consists of a fixed number of butterfly-groups. There are two popular schemes for FFT with SWP.

The first scheme (SWP-1) is the SWP *with butterfly-groups*. SWP-1 is implemented as a 3-level loop-nest. The outmost handles stages, the inner one handles groups, and the innermost loop handles different butterflies in each butterfly-group. Different butterflies do not depend on each other so that the innermost loop can be easily software-pipelined. In addition, the memory locations of different butterflies are next to each other, so that the addresses can be easily generated [16].

The second scheme (SWP-2) is the SWP *without butterfly-groups*. the SWP-2 is implemented as a 2-level loop-nest. The innermost one handles *all* butterflies in each stage. With SWP-2 the addresses for different butterflies need to be generated with bit-and, shift and addition operations. The address generation schemes has been maturely studied [17].

## III. Generic Multi-Phase Software-Pipelined PFFT on ILP Architecture

### A. Principles of The Design

The input and output-pruning in the FFT/IFFT are dual problems and they can be solved in a similar way [1][4-8]. In this paper we focus on output-pruning because it is important for modern wireless baseband [2] [1].

Our goal is to delivery full-flexibility and high-efficiency *at the same time*. In another word, the work is expected to efficient enough while being applicable for arbitrary cases. Our design exploits the opportunities and constraints in FFT algorithms, ILP architectures and their interactions.

First of all, the abundant flexibility of FFT algorithms is exploited. As we will show later, exploring FFT dataflow variants enables aggressive optimizations in subsequent steps.

Second, we introduce heterogeneous control flow structures. Specifically, different stages in the PFFT are partitioned into 3 phases, and different control flow structures are introduced

TABLE I
ADVANTAGES AND DISADVANTAGES OF SWP-1 AND SWP-2

| | Advantages | Disadvantages |
|---|---|---|
| SWP-1 | Minimal overhead for address-generation, minimal twiddle-factor access | Very high SWP overhead (prologue and epilogue) when group-size is small |
| SWP-2 | Minimal SWP overhead (prologue and epilogue) | High address-generation overhead, redundant twiddle-factor access |

in the 3 phases. This is to preserve SWP-efficiency well and minimize the control-overhead.

Third, based on the detailed analysis of different SWP schemes, we apply heterogeneous SWP schemes in the 3 phases. The advantages and disadvantages of different SWP schemes are summarized in Talble.I. Elaborately combining multiple SWP schemes will leverage their advantages while avoiding their negative sides.

Clearly, heterogeneities are introduced in our design. For signal processing ASICs, regular and homogeneous structures are preferred because higher structure-complexity incurs significantly increased area and design-time. However, in programmable implementations the structure-complexity can be easily implemented at low expenses. With appropriate optimizations, the cost of the structure-complexity is just increased code-size and slightly increased instruction-cache misses. We will prove this in experiments with detailed profiling statistics.

### B. The Exploration of Dataflow Variants

For PFFT with partial outputs, we explored different FFT dataflow variants and selected the DTSO (Decimation-in-Time Shuffling-on-Output) FFT as the dataflow structure. The example shown in Fig.1 is a DTSO PFFT. This variant is favored because of the following characteristics: **(1)** All butterflies in the same butterfly-group share the same twiddle factor. Hence, when applying the SWP-1 (with butterfly-group), only entire butterfly-group need only one memory-loading for twiddle-factor. **(2)** A significant part of the bit-reversal operation can be pruned because it is on the output. **(3)** We can structure the different stages into three phases, then apply heterogeneous control structures and SWP schemes in different phases, resulted in aggressive optimizations of cycle-counts, number of instructions, memory accesses and so on.

### C. SWP and Control Structures in Multiple-Phase PFFT

*1) Overview:* In the PFFT we can fully exploit the characteristics of the ILP and the SWP. With heterogenous control structures, SWP schemes and address-generation schemes, we can achieve efficient memory access, minimized numbers of instructions and maximized utilization of hardware-resources. We propose the multiple-phase partitioning to bring-in and organize the heterogeneity.

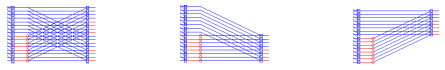
In Fig.1 the concept of multiple-phase partitioning is illustrated with a simple example. Note that in practical PFFT applications the size and irregularity might be far beyond the small example. In Fig.1, 7 stages of the DTSO PFFT are partitioned into three phases: The phase-1 (stage 1,2,3) consists of stages that are exactly the same as full FFT; The phase-2 (stage 4,5) consists of stages containing large butterfly groups; The phase-3 (stage 6,7) consists of stages containing very small butterfly groups.

With the partitioning, heterogenous control structures, SWP schemes and address-generation schemes can be introduced.

*2) The phase-1:* The phase 1 of the PFFT is exactly the same as that in a full FFT, so that *no extra* control structure is needed. Moreover, in phase-1 the size of butterfly-groups is large enough to enable a long loop-body for the SWP-1 (with butterfly-group). Comparing to the SWP-2, the SWP-1 brings minimized number of instructions and minimized twiddle-factor accesses. Note that the optimization is enabled because we select DTSO PFFT. With other FFT variants we may have to apply the SWP-2 to reduce the SWP overheads, at the expense of significant memory and address generation overhead.

*3) The phase-2:* The left-boundary of the phase-2 is then fixed because of the phase-1. The right-boundary of the phase-2 is determined to ensure that the size of butterfly-groups in phase-2 is still large enough to fill-in the parallel FUs (Function Units) in the given ILP architecture. Hence, the SWP-1 is applied in phase-2 as well. This again results in minimal overhead for address-generations and loadings of twiddle-factors.

In addition, the PFFT phase-2 requires extra control structures. In order to minimize the control-overhead, the control operations is on the granularity of butterfly-groups. In addition, instead of always processing full butterfly-groups, we differentiate the groups as that shown in Fig.3. These groups have different innermost loop implementations. In this way, significant amount of redundant computation and non-computation operations can be eliminated. Note that the SWP is *not* degraded by the control flow at all, the overhead of control flow is minimized as well.



(a) (b) (c)

Fig. 3. Different butterfly-groups in Phase 2. (a): Full butterfly-group. (b): Upper-half butterfly-group. (c): Lower-half butterfly-group. Differentiating the groups is to eliminate redundant computation and memory-access operations as much as possible.

*4) The phase-3:* The phase-3 is very different. As shown in Fig.1, the total number of butterflies in the phase-3 is much smaller than that in phase-1 and phase-2. In addition, the size of butterfly-groups in the phase-3 becomes very small. The total number of butterfly-groups is much larger.

According to the above observations, the SWP-1 will incur large overhead because groups are small and number of groups is large. Hence, we apply the SWP-2 (without butterfly-group). Each stage needs only one prologue and one epilogue for SWP-2, so that the overhead of SWP is minimized in the phase-3. In this phase the control-operation is on individual butterflies. Since the number of butterflies in the phase-3 is small, the total overhead for SWP-2 is still relatively low when comparing to the entire PFFT.

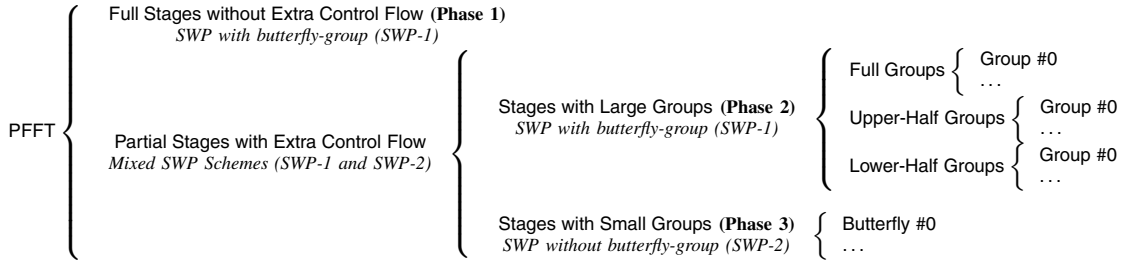



Fig. 2. The ILP-Friendly Software-Pipelined PFFT Described with Hierarchical Tree Structure

The above phase-partitioning with associated SWP schemes are summarized as a tree-structure in Fig.2. By introducing heterogeneity, the control-overhead is minimized. In addition, the advantages of different SWP schemes are well combined whereas the individual disadvantages are avoided.

TABLE II
ENCODED INFORMATION FOR MULTI-PHASE PFFT

| Phase | Encoded Information |
|---|---|
| 1 | The right-boundary of the phase-1 |
| 2 | The total number of groups, offsets of individual group |
| 3 | The total number of butterflies, offsets of individual butterfly |

### D. Encoding Scheme for Dataflow Information

We encode the dataflow information in a very compact table to enable full-flexibility. The encoding scheme is summarized in Table.II. The entire phase-1 is described with only one element. The elements (butterfly-groups or butterflies) in the phase-2 and the phase-3 are described with total numbers and the offsets of individual elements to the previous one.

With this scheme, for the stage 4 (the first stage in phase 2) in Fig.2, there are 4 full butterfly-groups: group 1,2,4,5, so that the encoded information associated with them is $\{4, 0, 1, 2, 1\}$, where $'4'$ is the total number of full butterfly-groups, $'0'$ is the offset of the first group, $'1'(2-1)$ is the offset of the second group to first group, $'2'(4-2)$ is the offset of the third to second, and $'1'(5-4)$ is the offset of the fourth to third. Similarly, the information for upper-half groups is $\{3, 2, 4, 1\}$. The encoding scheme for the phase 3 is similar to phase 2, except that the information in phase 3 describes butterflies but not butterfly-groups.

The encoding scheme results in a highly compressed table: The phase 1 is described with only one element; The phase 2 is compactly encoded because the description is on large butterfly-groups containing many operations; The phase 3 is on butterflies but the total number of butterflies is small. The compact table is also an advantage of the heterogeneous control structure. In the next section, we will show that the control-overhead is relatively small because the PFFT significantly reduces cycle-counts, instructions, memory accesses and so on.

## IV. EXPERIMENTS AND ANALYSIS

### A. Benchmarks Set

We extracted 10 representative benchmarks from practical baseband signal processing functionalities, include OFDMA demodulation and the multi-resolution broad-band spectrum scanning. The benchmarks are summarized in Table III.

The benchmarks cover a wide variety of irregular PFFT dataflow patterns. In previous work, the studied PFFTs were often with regularly-spaced (comb) output or block-output with size being $2^k$. However, practical applications often have much more complex scenarios. For instance, in the specification of 3GPP LTE, the size of sub-carrier resource block is an odd-number 25, *not* $2^k$. There might be multiple size-25 blocks dispersed over available sub-carriers. Moreover, for comb output in high-mobility 3GPP LTE demodulation the spacing is *not* $2^k$ neither. The output bins might consist of many small-size clusters. These practical settings incurs highly irregular dataflow in PFFT. It is very important to include these practical scenarios in the benchmark set.

In order to evaluate the maximum potentials for the benchmarks, we plot the abstract-complexity reduction-rate in Fig.4. The reduction-rate of complex-multiplications, complex-additions and shuffling-distances are plotted. The shuffling-distance is defined as the sum of memory-location distances in all shuffling-involved operations, such as bit-reversal and butterflies.

TABLE III
SUMMARY OF BENCHMARKS

| ID | Baseband Func. | FFT Size | Output | Indexes of Output Points | Regularity |
|---|---|---|---|---|---|
| 1 | Spe.Scan | 1024 | 128 | {1,9,..,1017} | High |
| 2 | Spe.Scan | 2048 | 64 | {256,257,..,383} | High |
| 3 | Spectrum Scan | 2048 | 64 | {1, 2, ..., 32} and {256,260,..,508} | Low |
| 4 | OFDMA | 512 | 25 | {51, 52, ..., 75} | Low |
| 5 | OFDMA | 1024 | 50 | {51, 52, ..., 75} and {151, 152, ..., 175} | Very low |
| 6 | OFDMA | 1024 | 125 | {900, 901, ..., 1024} | Low |
| 7 | OFDMA | 512 | 25 | {26, 31, ..., 146} | Low |
| 8 | OFDMA | 1024 | 50 | {26, 31, ..., 271} | Low |
| 9 | OFDMA | 1024 | 50 | {26, 51, ..., 251}, {27, 52, ..., 252}, {28, 53, ..., 253}, {29, 54, ..., 254} and {30, 55, ..., 255} | Very low |
| 10 | OFDMA | 1024 | 50 | {1, 11, ..., 241} and {775, 715, ..., 1015} | High |

### B. Implementations and References

In this paper we include the results on VLIW, one of the most important instances of ILP architectures. Since the TMS320C6000 family is often used to benchmark loop optimization and compilation techniques for (clustered) VLIW [18] [19], we give *reproducible* experiment results on the commercially-available and accessible TMS320C6713. TMS320C6713 is a typical VLIW DSP supporting 8 32-bit instructions per-cycle. The 8 FUs are organized as two clusters. The Level-1 memory includes 4K-byte direct-mapped instruction cache (L1P) and 4K-byte 2-way data cache (L1D).

Because the memory access of our work is highly deterministic and predictable, Level-2 memory of the DSP is configured as non-cache and controlled by software. In order to gain enough insights into the proposed scheme, we perform extensive profiling on fine-grain details of executions, including (1) cycle-count, (2) number of instructions, (3) NOP (No Operation) instruction, (4) L1D access/hit/miss and (5) L1P access/hit/miss.

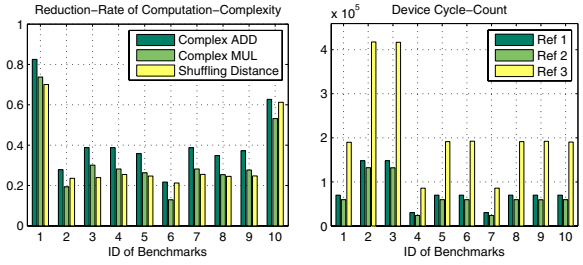
The proposal is implemented in C code and iteratively optimized according to the feedback from compilation and profiling, ensuring a satisfactory SWP efficiency on the given ILP architecture.

Aiming at fair comparisons, we implemented three references in our work: The first (Ref-1) is the full FFT using the SWP-1 (with butterfly-group). The second (Ref-2) is the full FFT using the SWP-2. The third (Ref-3) is the proposed "General FFT Pruning" implementation in [6].

### C. Detailed Comparison and Analysis on Statistics

*1) Comparisons to The Existing PFFT Scheme (Ref-3):* First we compare the proposed "General FFT Pruning" implementation (Ref3) to full FFT implementations. The Ref-3 is based on a $N \times log_2 N$ table with dataflow information encoded. From 4 (b) we can observe that the proposed PFFT in [6] does *not* reduce cycle-counts at all. The key reason is that the irregular dataflow and extensive conditions in the innermost loop are extremely inefficient on ILP architectures. This clearly indicates that elaborate optimizations are *necessary* when mapping the irregular dataflow on to parallel architectures.

In the following, we will compare the proposed scheme to the Ref-1 and the Ref-2. Detailed profiling statistics will be compared and thoroughly analyzed.




(a) (b)

Fig. 4. Benchmarks and References. (a) Complexity reduction in benchmarks(b) Cycle-count of all references

*2) Cycle Count and Number of Instructions:* Fig.5 (a) plots the cycle-count. We can observe that the proposed scheme achieves significant cycle-count reductions for all of the benchmarks. Comparing to the Ref1, the PFFT achieves 32.5% to 87.5% cycle-count reductions; comparing to Ref2, the PFFT achieves 20.5% to 85.4% cycle-count reductions. Surprisingly, the reduction-ratio of cycle-count is even higher than the reduction-ratio of abstract-complexity shown in Fig. 4. The reasons behind this can be found out in Fig.5 (b) and (c).

As discussed before, the SWP-1 in the Ref-1 incurs minimal overheads for address generations, but it suffers from the a low SWP efficiency when the size of butterfly-groups is too small because prologues and epilogues of the SWP will be dominant. On the contrary, the SWP-2 in the Ref-2 significantly reduces the overhead for setting-up SWP, but suffers from large overhead for address generations and redundant twiddle-factor access. This has been verified in experiments. In Fig.5 (b) the total number of instructions in the Ref-2 is significantly larger than the Ref-1. Because many FUs are idle in the prologues/epilogues and SWP-1 needs more prologues/epilogues, the number of NOP instructions in the Ref-1 is significantly larger than the Ref-2,

Although both the SWP-1 and the SWP-2 have disadvantages, the proposed scheme combines the advantages of different SWP schemes but the disadvantages are avoided. In the phase-1 and phase-2 the SWP-1 is applied because group is large enough. In phase-3 SWP-2 is applied because group is small and number of group is large. The heterogeneous SWP schemes in significantly reduce the total number of instructions and cycle-count comparing to both Ref-1 and Ref-2. Specifically, the PFFT reduces 11.2% to 74.8% instructions comparing to the Ref1, reduces 55.9% to 86.5% instructions comparing to the Ref2. The cycle-count reduction-rate of the proposed PFFT is even higher than that of abstract-complexities, because the disadvantages of both SWP schemes are elaborately avoided. The reduction in cycle-count and overhead clearly prove that the overhead in our scheme is very small when comparing to the large gain. Full-flexibility and high-efficiency have been delivered at the same time with our proposal.

*3) Cache Access and Hit/Miss:* The reductions of cache accesses are presented in Fig.5 (d)(e). On L1D, the proposal reduces 16.1% to 78.1% accesses comparing to the Ref-1, reduces 21.2% to 79.4% accesses comparing to the Ref-2. The SWP-2 loads twiddle-factor from data memory for *each* butterfly, whereas the SWP-1 in Ref-1 loads twiddle factor only once for the entire butterfly-group. Hence, Ref-2 incurs more L1D accesses than Ref1. Again, with increased structure-complexity the proposed PFFT elaborately combines the advantages of different SWP schemes.

On L1P we have similar observations. SWP-2 incurs large overheads for address-generations and memory-accesses so that more instructions and L1P accesses are needed. The heterogenous control structures and SWP schemes in the proposal minimize the overhead for both address generations and control operations, bringing 19.5% to 78.5% reduction comparing to the Ref1 and 51.1% to 87.1% reduction comparing to the Ref2.

The L1 cache hit rate is plotted in Fig.5 (f)(g). We can observe that the hit rate is very high. However, the proposal incurs slight hit-rate degradations for some of the benchmarks. This is a natural phenomenon, because PFFT pruned some of the computations so that the accesses on the same piece of data are less than that in a full FFT. Again, the reductions in L1D and L1P accesses clearly prove that the overhead in our

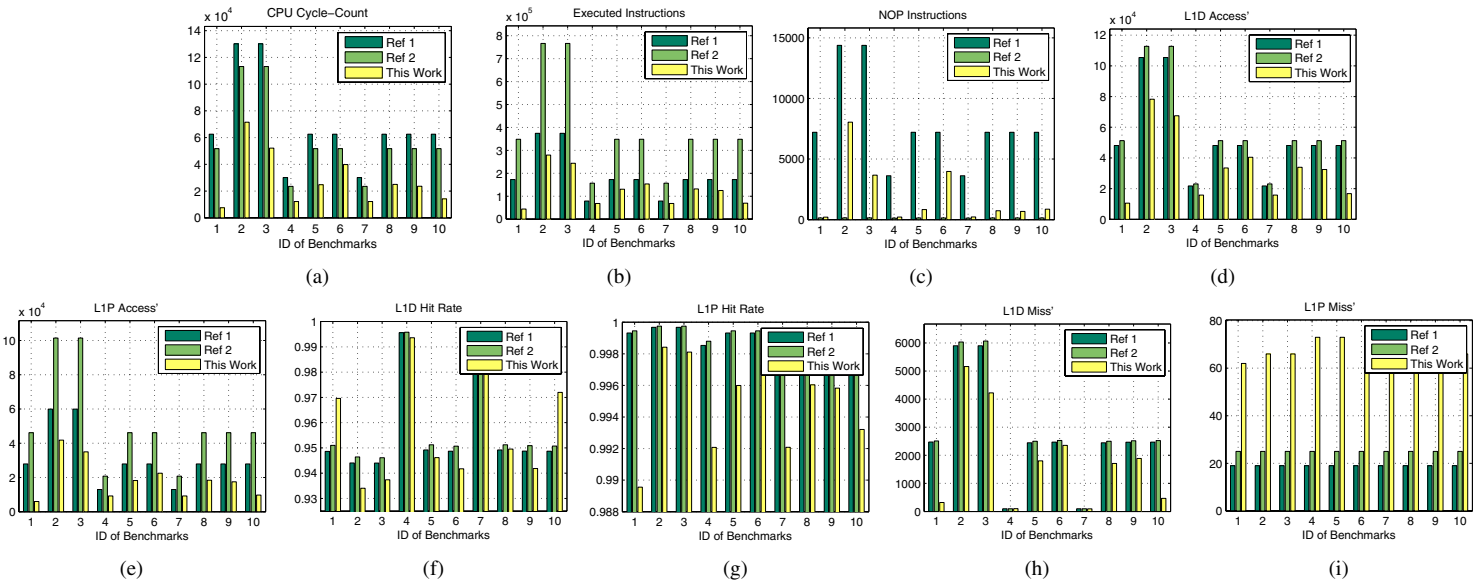



Fig. 5. Detailed Comparison of Profiled Statistics on The Architecture

scheme is very small comparing to the large gain.

The absolute numbers of L1 cache misses are plotted in Fig.5 (h)(i). These figures are very important because the accesses to large memory in hierarchy are energy-hungry operations. We can observe that the proposed PFFT brings reduction in L1D misses as well. However, in (i) we observe that the number of L1P misses significantly increases. In the proposal the increased heterogeneity and structure-complexity unavoidably incur penalties in L1P misses. Fortunately, the number of increased misses in L1P is very small when comparing to the reduced L1D misses. Taking into account the significant reductions in combined L1P and L1D accesses, we can safely conclude that the energy consumption in memory sub-system is aggressively optimized with our proposal.

## V. Conclusion

Although the theoretical aspects of PFFT have been intensively studied in past decades, efficient implementations were rarely reported. We proposed an generic scheme to software-pipeline the irregular dataflow of arbitrary PFFT onto ILP architectures. The proposal achieves significant reductions in cycle-counts, instructions, L1D/L1P accesses and misses. The reduction-rate of cycle-counts is even larger than that of the abstract complexity-reductions. Full-flexibility and high-efficiency have been delivered at the same time.

In the deep-sub micron era, programmable-architecture based systems become more and more popular for the sake of time-to-market and non-recurring-engineering-cost issues. In this context, we advocate the design-flow which performs architecture-aware explorations and transformations from the very beginning of the design-flow, so that the constaints/opputunity of programmable-architecture can be better exploited.

## References


[1] Z.Hu and H. Wan, "A novel generic fast Fourier transform pruning technique and complexity analysis," *IEEE Trans. Signal Process.*, Jan. 2005 Volume: 53, page(s): 274- 282.

[2] Murphy, C.D. "Low-complexity FFT structures for OFDM transceivers", *IEEE Trans. on Signal Process.*, Volume: 50, Issue: 12 On page(s): 1878-1881, Dec 2002.

[3] 3GPP LTE TR 25.814: Physical layer aspects for E-UTRA.

[4] J. D. Markel, FFT pruning, *IEEE Trans. Audio Electroacoust.*, vol. 19, pp. 305-311, Dec. 1971.

[5] S. S. He and M. Torkelson, Computing partial DFT for comb spectrum evaluation, *IEEE Signal Process. Lett.*, vol. 3, pp. 173-175, Jun. 1996.

[6] Alves, R.G. Osorio, P.L. Swamy, M.N.S. General FFT pruning algorithm, the 43rd IEEE Midwest Symposium on Circuits and Systems, 2000

[7] S. R. Rangarajan and S. Srinivasan, Generalized method for pruning an FFT type of transform, *Proc. Inst. Elect. Eng. Vis. Image Signal*, vol. 144, pp. 189-192, 1997.

[8] H. V. Sorensen and C. S. Burrus, Efficient computation of the DFT with only a subset of input or output points, *IEEE Trans. Signal Process.*, vol. 41, pp. 1184-1200, Mar. 1993.

[9] K. Nagai, Pruning the decimation-in-time FFT Algorithm with frequency shift, *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-34, pp. 1008-1010, Aug, 1986.

[10] Shousheng He, Torkelson, M. VLSI computation of the partial DFT for (de)modulation inmulti-channel OFDM system, *IEEE PIMRC 1995*. Sep 1995

[11] R.Rajbanshi, A. M. Wyglinski, and G. J. Minden, An Efficient Implementation of NCOFDM Transceivers for Cognitive Radios, *IEEE CrownCom 2006*.

[12] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti and K. Flautner , SODA: A High-Performance DSP Architecture for Software-Defined Radio, *IEEE Micro 2007*

[13] Ta. Kumura, M. Ikakawa et al, VLIW DSP for mobile Applications, *IEEE Signal processing magazine July 2002*

[14] B. Mei, S. Vernalde, D. Verkest, R. Lauwereins, Design Methodology for a Tightly Coupled VLIW/Reconfigurable Matrix Architecture: A Case Study, *Proc. of DATE 2004*, pp. 1224-1229,

[15] V. Allan, R. Jones, R. Lee, and S. Allan. Software Pipelining. *ACM Computing Surveys*, 27(3), September 1995.

[16] TMS320C64x DSP Library Programmer's Reference (Rev. B)

[17] Yutai Ma, "An Effective Memory Addressing Scheme for FFT Processors," *IEEE Trans. Signal Process.*, vol. 47, Issue 3, pp. 907–911, March 1999

[18] Y. Qian, S. Carr and P. Sweany. "Loop Fusion for Clustered VLIW Architectures", In Proceedings of the ACM 2002 Joint Conference on Languages, Compilers and Tools for Embedded Systems and Software and Compilers for Embedded Systems, Berlin, Germany, June 19-21, 2002.

[19] Z. Shao, C. Xue, Q. Zhuge, B. Xiao and E. H.-M. Sha, Loop Scheduling with Timing and Switching-Activity Minimization for VLIW DSP, *ACM Transactions on Design Automation of Electronic Systems*, vol. 11, no. 1, pp. 165-185, Jan. 2006.