Selective Spanning with Fast Enumeration: A Near Maximum-Likelihood MIMO Detector Designed for Parallel Programmable Baseband Architectures

Min Li, Bruno Bougard, Eduardo Lopez, Andre Bourdoux, David Novo, Liesbet van Der Perre, Francky Catthoor

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the ICC 2008 proceedings.

Selective Spanning with Fast Enumeration: A Near Maximum-Likelihood MIMO Detector Designed for Parallel Programmable Baseband Architectures

Min Li†‡, Bruno Bougard†, Eduardo Estraviz Lopez†, Andre Bourdoux †,
David Novo†‡, Liesbet Van Der Perre†, Francky Catthoor†‡
† Nomadic Embedded System Division, IMEC, Leuven, Belgium
‡ ESAT, K.U.Leuven, Leuven, Belgium
Email: {limin, bougardb, lopezest, bourdoux, novo, vdperre, catthoor}@imec.be

***Abstract*—ML and near-ML MIMO detectors have attracted a lot of interest in recent years. However, almost all of the reported implementations are delivered in ASIC or FPGA. Our contribution is to co-optimize the near-ML MIMO detector algorithm and implementation for parallel programmable baseband architectures, such as DSPs with VLIW, SIMD or vector processing features. Although for hardware the architecture can be tuned to fit algorithms, for programmable platforms the algorithm must be elaborately designed to fit the given architecture, so that efficient resource-utilizations can be achieved. By thoroughly analyzing and exploiting the interaction between algorithms and architectures, we propose the SSFE (Selective Spanning with Fast Enumeration) as an architecture-friendly near-ML MIMO detector. The SSFE has a distributed and greedy algorithmic structure that brings a *completely deterministic and regular dataflow*. The SSFE has been evaluated for coded OFDM transmissions over 802.11n channels and 3GPP channels. Under the same performance constraints, the complexity of the SSFE is significantly lower than the K-Best, the most popular detector implemented in hardware. More importantly, SSFE can be easily parallelized and efficiently mapped on programmable baseband architectures. With TI TMS320C6416, the SSFE delivers 37.4 - 125.3 Mbps throughput for 4x4 64QAM transmissions. To the best of our knowledge, this is the first reported near-ML MIMO detector explicitly designed for parallel programmable architectures and demonstrated on a real-life platform.**

I. Introduction

Although the Moore's Law predicted a fast evolution of the semiconductor integration, the increment of silicon-capability has been rapidly exhausted by the explosion of signal processing complexity in wireless communications [1]. When applying MIMO (Multiple Input Multiple Output) transmissions, the remarkable throughput improvement comes at the cost of significantly increased receiver complexity.

With SDM (Spatial Division Multiplexing) transmissions, the major complexity-increment is in the MIMO detector. Among existing MIMO detectors, the ML (Maximum-Likelihood) and near-ML detectors are superior to traditional linear detectors. In recent years, the algorithmic optimizations and implementations of ML/nearl-ML detector have attracted lots of interest. However, almost all of the implementations are delivered in ASIC (Application Specific Integrated Circuit) or FPGA (Field Programmable Gate Array) [2-10].

Being different from the existing work, our contribution is to co-optimize the algorithm and implementation for scalable near-ML MIMO detector on *parallel programmable baseband architectures*, such as the DSPs (Digital Signal Processors) with VLIW (Very Long Instruction Word), SIMD (Single Instruction Multiple Data) or vector processing features.

The work is in the context of future baseband for SDR (Software Defined Radio). With the exploding design and processing cost in the deep sub-micron era, the current trend is to implement as much possible baseband functionalities on programmable or reconfigurable platforms. The SDR paradigm, which was mainly successful in the base-station and military segment, is currently emerging also in the handset market. Recently, tremendous research efforts have been investigated in both the industry and the academia for parallel programmable baseband architectures targeting mobile terminals [11] [12].

Unfortunately, none of the existing near-ML detectors fit programmable architectures well. Sphere decoders [3] [4] and most of its variants [5] [6] are essentially sequential and non-deterministic, so that the parallelization is difficult. On the other hand, although the K-Best, QRD-M and their variants [2] [7] [9] have been realized in hardware implementations, they have a fundamental problems when mapping on parallel programmable architectures: The spanning-sorting-deleting process incurs irregular dataflow, non-deterministic control flow, extensive shuffling and extensive memory-rearrangement. These characteristics will result in very low resource-utilizations on parallel programmable architectures.

In order to bridge the algorithm-architecture gap for near-ML MIMO detectors, we propose the SSFE (Selective Spanning with Fast Enumeration) as novel detector with explicit architecture-friendliness. The SSFE has a distributed and greedy algorithmic structure that enables efficient heuristics to solve decomposed problems locally. Comparing to the K-Best, the SSFE not only significantly reduces the algorithmic complexity, but also results in a completely *deterministic* and *regular* dataflow in the algorithm. Hence, the SSFE can be easily parallelized and efficiently mapped on programmable architectures. This has been demonstrated with reproducible results on TI TMS320C6416, a real-life commercial fixed

point VLIW DSP.

The remaining part of the paper consists of the following sections: Section II introduces the context and related work; Section III presents the details of the proposed architecture-friendly SSFE; Section IV presents the performance evaluation and implementation results on TI TMS320C6416 fixed point VLIW DSP; Section VI concludes the paper and briefs the future work.

II. Background

Consider a MIMO system where Nt different signals are transmitted and arrive at an array of Nr ($Nt \leq Nr$) receivers via a flat-fading channel. The system can be viewed as transmitting an $Nt \times 1$ vector signal $\mathbf{s}$ through an $Nr \times Nt$ matrix channel $\mathbf{H}$, with $Nr \times 1$ Gaussian noise vector $\mathbf{n}$ added at the received vector signal $\mathbf{y}$: $\mathbf{y} = \mathbf{Hs} + \mathbf{n}$. With OFDM (Orthogonal Frequency Division Multiplexing) transmissions such as that in IEEE 802.11n and 3GPP LTE (Long Term Evolution), frequency-selective channels are converted to a set of parallel flat-fading channels.

The MIMO detector is designed to recover $\mathbf{s}$ from the received signal $\mathbf{y}$. Popular schemes include linear detection, SIC (Successive Interference Cancelation) and ML/Near-ML detectors. Extensive surveys can be found in [13] [14].

The ML detection is defined as

$$\hat{s} = \arg \min_{s \in \Omega^{Nt}}, \|\mathbf{y} - \mathbf{Hs}\|^2 \tag{1}$$

where Ω^{Nt} is the set containing all the possibilities of $Nt \times 1$ vector signal $\mathbf{s}$.

SD (Sphere Decoding) [3-6] solves the ML detection problem by applying the QRD (Orthogonal-Triangular-Decomposition) to the channel matrix: $\mathbf{H} = \mathbf{QR}$, where $\mathbf{Q}$ is a orthogonal matrix and $\mathbf{R}$ is an upper triangular matrix. It can be shown [5] that

$$||\mathbf{y} - \mathbf{Hs}||^2 = c + ||\hat{\mathbf{y}} - \mathbf{Rs}||^2, \;\; \hat{\mathbf{y}} = \mathbf{Q}^H\mathbf{y} \tag{2}$$

where c is a constant.

We can construct a spanning-tree to solve Eq.(1). Level of the tree is $Nt+1$; mark the root-level as $i = Nt+1$ and the leaf-level as $i = 1$. Each node at level $i \in \{2, .., Nt+1\}$ is expanded to $\mathcal{C}$ nodes at level $i+1$, where $\mathcal{C}$ is the constellation size. In this tree each node at level $i \in \{1, 2, .., Nt\}$) is uniquely described by the partial vector symbols $\mathbf{s}^i = [s_i, s_{i+1}, .., s_{Nt}]$, the leaves at level $i = 1$ correspond to all possible vector-symbols Ω^{Nt}.

Annotate the root node with $T_{Nt+1} = 0$ and starting from Level $i = Nt$, the PED (Partial Euclidean Distance) of partial symbol vector $\mathbf{s}^i = [s_i, s_{i+1}, .., s_{Nt}]$ is $T_i(\mathbf{s}^i) = T_{i+1}(\mathbf{s}^{i+1}) + ||e_i(\mathbf{s}^i)||^2$, where the PED-increment $||e_i(\mathbf{s}^i)||^2$ is

$$||e_i(\mathbf{s}^i)||^2 = ||\hat{y}_i - \sum_{j=i}^{Nt} R_{ij}s_j||^2 \tag{3}$$

$||e_i(\mathbf{s}^i)||^2$ is obviously non-negative, so that the PED increases monotonically from root to leaves. Hence, formulation in Eq.(2) has now been transformed as a tree-search which finds the leaf at level $i = 1$ with the minimal PED $T_1(\mathbf{s}^1)$. The leaf with the minimal PED corresponds to the $\hat{\mathbf{s}}$ in Eq.(1).

Various depth-first searching algorithms have been proposed for SD [3-6]. However, most of these algorithms are essentially depth-first serial tree-search. In addition, most of them have the non-deterministic dynamism depending on the channel matrix and the SNR. Hence, they are not suited for parallel programmable architectures.

Instead of performing depth-first searching. The sub-optimal K-Best (similar to QRD-M) and its variants perform breadth-first searching [2] [7] . The K-Best and variants are mostly ASIC-minded algorithms. They keep K best nodes on each level of the tree. When going from level $i+1$ to i, K-Best first spans the K nodes at level $i+1$ to $K\mathcal{C}$ nodes, where $\mathcal{C}$ is the constellation size. After spanning, K-Best sorts the $K\mathcal{C}$ nodes, the K best nodes are selected and the rest nodes are deleted. Both strict sorting [7] and approximating sorting [2] have been proposed. The spanning-sorting-deleting process are repeated for Nt times until reaching the leaf nodes. Clearly, K-Best involves modular and repetitive operations that are easily parallelized in hardware. Hence, it has became the most popular near-ML detector in ASICs and FPGAs.

Although the K-Best suits parallel VLSI architecture well, it has many problems on parallel programmable architecture: (1) extensive shuffling operations have to be decomposed into supported shuffling instructions, which incurs significant cycle and energy overhead on programmable architectures; (2) the execution is not deterministic and regular, data-dependent memory-operations and computations will significantly degrade the resource-utilizations on programmable architectures; (3) the memory arrangement incur cycle-count and energy overhead in both datapath and memory; (4) the complexity of the spanning-sorting-deleting process is still too high. Although [10] uses the SD to benchmark VLIW processor, the aforementioned problems are not solved yet.

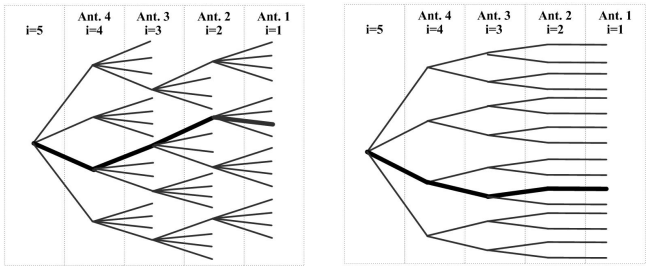
Clearly, further innovations are necessary when implementing near-ML detectors on programmable architectures. We will substitute the spanning-sorting-deleting process with architecture-friendly dataflow structures and procedures.

III. SSFE for Parallel Programmable Architecture

A. Overview

We propose the SSFE as a novel near-ML detector with explicit architecture-friendliness. The key feature of the SSFE is the completely deterministic and regular dataflow structure. Abundant vector-parallelism is enabled in the SSFE; memory rearrangement, shuffling operations and non-deterministic dynamism are all eliminated. Experiment results show that these characteristics will bring highly-efficient resource-utilizations on real-life parallel programmable architectures.

The SSFE can be uniquely characterized by a vector $\mathbf{m} = [m_1, \ldots, m_{Nt}]$. Starting from level $i = Nt$, SSFE spans each node at level $i+1$ to m_i nodes at level i. The spanned nodes are never deleted. Hence, the total number of nodes at level i is $\prod_{k=i}^{Nt} m_k$. If the node at level $i = Nt+1$ has the associated



(a) (b)

Fig. 1. Topology of Trees in K-Best and SSFE.(a) K-Best; (b)SSFE

partial symbol vector being $\mathbf{s}^{i+1} = [s_{i+1}, .., s_{Nt}]$, the spanning is to select a set of $\mathbf{s}^i = [s_i, s_{i+1}, .., s_{Nt}]$ in the way that $||e_i(\mathbf{s}^i)||^2$ is minimized.

The topology of search-trees in the K-Best and the SSFE are compared in Fig.1, where (a) is for K-Best with $K = 4$. (b) is the for SSFE with $\mathbf{m} = [1, 2, 2, 4]$. The transmission is 4x4 QPSK. The bold lines mark the path and leaf with the minimal PED. Note that (a) is just one possibility of the K-Best execution, which is essentially dynamic. On the contrary, the SSFE brings completely deterministic and regular dataflow structures.

Essentially, the SSFE is a distributed and greedy algorithm similar to the dynamic programming. It is greedy because it minimizes $||e_i(\mathbf{s}^i)||^2$ at each level of the tree. In addition, it is distributed because the minimization of $||e_i(\mathbf{s}^i)||^2$ is *local* for *each* node at level $i+1$ when spanning this node to m_i nodes at level i. On the contrary, in the K-Best, the spanning-sorting-deleting process is based on the PED ($T_i(\mathbf{s}^i)$) of $K\mathcal{C}$ spanned nodes at level i. Hence, in the K-Best the sorting of $T_i(\mathbf{s}^i)$ is performed *globally* on $K\mathcal{C}$ nodes, which are spanned from K different nodes at level $i+1$. Clearly, when $K\mathcal{C}$ is large huge amount of operations are required. In contrast to the K-Best, the distributed and greedy algorithmic structure of SSFE enables very efficient heuristics (Fast Enumeration) to minimize $||e_i(\mathbf{s}^i)||^2$. Note that the SSFE never deletes any nodes or paths. On the contrary, at level $i = Nt$ the K-Best needs to delete $K\mathcal{C} - K$ nodes and associated paths. Comparing to the K-Best, the SSFE involves no memory rearrangement.

Our scheme is also different from the fixed-complexity detector in [15], despite the similarities in the topology of the spanning-trees. First of all, in our scheme we select a set of $\mathbf{s}^i = [s_i, s_{i+1}, .., s_{Nt}]$ to minimize $||e_i(\mathbf{s}^i)||^2$ but not $T_i(\mathbf{s}^i)$. Second, sorting or deleting are *not* involved in the SSFE.

B. Fast Enumeration

To minimize $||e_i(\mathbf{s}^i)||^2$ during spanning, we may use the same technique as the K-Best. However, as we discussed before, the spanning-sorting-deleting process are not friendly to parallel programmable architectures.

Fortunately, the distributed and greedy algorithmic structure of the SSFE enable us to apply very simple heuristics to approximate the sorting operations. The heuristics is called FE (Fast Enumeration). To derive the FE, we first rewrite Eq.(3)

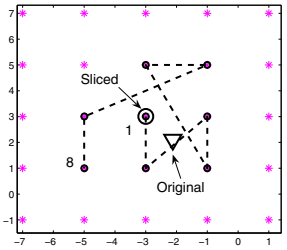


Fig. 2. Example of Fast Enumeration with 8 Constellation Points

as

$$\begin{aligned} ||e_i(\mathbf{s}^i)||^2 &= ||\hat{y}_i - \sum_{j=i}^{Nt} R_{ij}s_j||^2 \\ &= ||\underbrace{\hat{y}_i - \sum_{j=i+1}^{Nt} R_{ij}s_j}_{b_{i+1}(\mathbf{s}^{i+1})} - R_{ii}s_i||^2 \end{aligned} \quad (4)$$

Clearly, the minimization of $||e_i(\mathbf{s}^i)||^2$ is equivalent to the minimization of $||e_i(\mathbf{s}^i)/R_{ii}||^2$. Hence, from Eq.4 we derive

$$||e_i(\mathbf{s}^i)/R_{ii}||^2 = ||\underbrace{b_{i+1}(\mathbf{s}^{i+1})/R_{ii}}_{\xi_i} - s_i||^2 = ||\xi_i - s_i||^2 \quad (5)$$

Eq.5 gives the geometrical interpretation for minimizing $||e_i(\mathbf{s}^i)/R_{ii}||^2$. Specifically, minimizing Eq.5 is essentially to select the closest complex constellation point to ξ_i. For SSFE, the FE is to select a set of closest constellations around the point ξ_i.

When $m_i = 1$, the closest constellation to ξ_i is $p_1 = \mathcal{Q}(\xi_i)$, where $\mathcal{Q}$ is the slicing operator. When $m_i > 1$, more constellations can be efficiently enumerated based on the vector $d = \xi_i - \mathcal{Q}(\xi_i)$. For $m_i \leq 4$, the points can be enumerated in the following way (with C-like syntaxes):

$$\begin{aligned} \phi &= |\Re(d)| > |\Im(d)| \\ p_2 &= \mathcal{Q}(\xi_i) + 2(sgn(\Re(d))\phi + j(sgn(\Im(d))(!\phi))) \\ p_3 &= \mathcal{Q}(\xi_i) + 2(sgn(\Re(d))(!\phi) + j(sgn(\Im(d))\phi)) \\ p_4 &= \mathcal{Q}(\xi_i) + 2(sgn(\Re(d)) + j(sgn(\Im(d))) \end{aligned} \quad (6)$$

where '$sgn()$' is the operator for sign (positive/negtive), '!' is the logic-not operator like that in C.

Fundamentally, the technique applied here is to *incrementally* grow the set around ξ_i with efficient-heuristics based approximations. For example, if $|\Re(d)| > |\Im(d)|$, the second closest constellation (p_2) to ξ_i is on the horizontal-line where $\mathcal{Q}(\xi_i)$ stays, and the distance between $\mathcal{Q}(\xi_i)$ and p_2 is $2(sgn(\Re(d))$. If $|\Re(d)| < |\Im(d)|$, p_2 is on the vertical-line where $\mathcal{Q}(\xi_i)$ stays, and the distance is $2j(sgn(\Im(d))$. In order to avoid $if\text{-}then$ statements and to make a deterministic dataflow in the enumeration, we write the expressions of p_2 as that in Eq.(6). Similarly, p_3 and p_4 are enumerated with simple operations.

Following this way, $\{p_5, .., p_8\}$ can be enumerated as:

$$\begin{aligned} p_5 &= \mathcal{Q}(\xi_i) - 2j(sgn(\Im(d))),\ p_6 = p_4 - 4j(sgn(\Im(d))) \\ p_7 &= \mathcal{Q}(\xi_i) - 2sgn(\Re(d)),\ p_8 = p_4 - 4sgn(\Re(d)) \end{aligned} \quad (7)$$

Note that the $2\times$ and $4\times$ are 1-bit and 2-bit left-shift operations, respectively. For the sake of clarity, boundary conditions are ignored in the above formulations. In practical implementations the boundary conditions need to be examined.

An example of an 8-points enumeration is shown in Fig.2. The first point and the last point are annotated with numbers. We can enumerate more points in the same way. In practical implementations, we consider $m_i \in 1, 2, 4, 8, 16$ to simplify the address generation scheme on programmable architectures.

The FE has clear advantages over the PSK-like enumeration implemented in [5] [6]. The PSK-like enumeration depends on constellations size; it needs to examine the constellations on 1, 3, 9 co-centric circles for QPSK, 16QAM and 64QAM, respectively. When using 64QAM scheme the complexity is still very high. In addition, the PSK-like enumeration require trigonometric functions and a lot of divisions, which makes it difficult on most programmable platforms. On the contrary, the FE scheme in the SSFE is independent on constellation size, so that handling 64QAM will be as efficient as handling QPSK. More importantly, the FE is based on very simple and architecture-friendly operators such as addition, subtraction, bit-not and shift.

C. Parallelization of The SSFE

As shown in Fig.1, the dataflow in the SSFE is completely deterministic and regular. Hence, it is simple to parallelize the SSFE for parallel programmable architectures. There are at least two options. First, we can observe that the spanning operations of different nodes can be parallelized. Second, we can search multiple trees simultaneously. The second scheme is preferred because of the following advantages: (1) It brings abundant vector-parallelism, which can be easily mapped on VLIW, SIMD or vector architectures; (2) the parallelization is scalable, the number of parallel trees can be determined according to the supported parallelism on the given architecture; (3) this scheme perfectly fits OFDM and OFDMA systems, where the detection is essentially parallel for blocks of MIMO symbols.

IV. Experiment Results

A. Performance Evaluation

1) Setup: First we study the BER (Bit Error Rate) of the SSFE for OFDM and OFDMA systems. Coded transmissions over indoor/outdoor channels are evaluated. For indoor channels, we use the channel models specified in IEEE 802.11n. For outdoor channels, we use the channel models specified in 3GPP LTE. For the sake of limited space, herein we show only 1/2 coded hard-output 64QAM 4×4 transmissions. Fig.3, the BER of the ZF (Zero-Forcing) detector, the SD based ML detector, the K-Best and the SSFE are evaluated. The SSFE is plotted with solid lines, with $\mathbf{m}$ ranging from $[1, 1, 1, 1]$ to $[1, 2, 4, 16]$ (124F).

2) BER: As expected, the ML detector, the K-Best and the SSFE are all superior to the ZF detector. When $\mathbf{m} = [1, 2, 4, 16]$, the gap between the SSFE and the ML is very small. Specifically, at BER=10^{-4} the SSFE is only 0.1dB away from the ML detector. When comparing the K-Best to the SSFE, we can observe that the K-Best with $K = 4$ is close to the SSFE with $\mathbf{m} = [1, 1, 2, 4]$, and the K-Best with $K = 8$ is close to the SSFE with $\mathbf{m} = [1, 2, 4, 8]$.

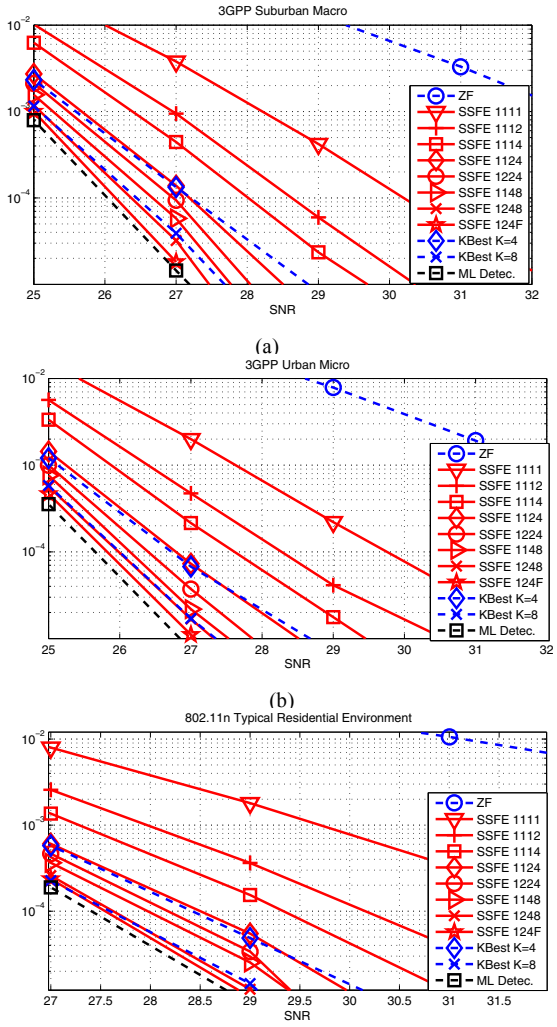


Fig. 3. BER Comparisons. (a) 3GPP Suburban Macro; (b) 3GPP Urban Micro; (c) 802.11n Typical Residential Environment;

3) Scalability: Fig.3 clearly shows that changing $\mathbf{m}$ brings the quality/cost scalability in the SSFE. This is very important for SDR. In practical communication systems the channels and data-rate are varying. In order to minimize the power consumptions in SDR, we can dynamically adjusting the $\mathbf{m}$ according to the aforementioned dynamism to perform just-enough processing.

Although the SSFE can approach the ML very closely with large $\mathbf{m}$, it is necessary for only very poor channels with low SNR or bad numerical properties. These poor-channel cases are important but rare. In most practical cases, a just-enough $\mathbf{m}$ is preferred to minimize power-consumptions. At BER=10^{-4}, the SSFE with $\mathbf{m} = [1, 1, 2, 4]$ is already 7dB better than the ZF. Hence, we will mainly study the the SSFE implementation with small $\mathbf{m}$.

TABLE I
COMPLEXITY COMPARISON BETWEEN SSFE AND K-BEST

	SSFE					K-Best		
	1111	1112	1114	1124	1248	K=2	K=4	K=8
CMUL	18	34	66	114	638	460	856	1648
CADD	26	51	101	177	1025	1120	3744	13600

B. Complexity

The associated complexities of the SSFE and the K-Best are compared in Table.I. The CMUL (Complexity Multiplications) and CADD (Complex Additions) are the (equivalent) operations required to detect one MIMO symbol in 4x4 64QAM transmissions. We exclude the complexity of the QRD preprocessing because it is very low in practical OFDM systems if appropriate optimizations are applied [16].

From Table.I we can observe that the SSFE has remarkable advantages of complexities. For instance, the SSFE with $\mathbf{m} = [1, 2, 4, 8]$ has a BER close to the K-Best with $K = 8$, while the SSFE has a significantly lower complexity.

C. Implementation Results on TI TMS320C6416

1) Setup: Besides the comparisons of abstract complexity, we have verified on a real-life platform that the SSFE is indeed very friendly to parallel programmable architectures. In this paper we include *reproducible* experiment results on the TI TMS320C6416 fixed point VLIW DSP. TMS320C6416 supports 8 32-bit instructions per-cycle, controlling 8 parallel FUs (Function Units). Level-1 memory consists of 16 K-Byte direct-mapped instruction cache (L1P) and 16 K-Byte 2-way set-associative data cache (L1D). Memory accesses of the SSFE is highly deterministic, Level-2 memory is configured as SRAM and controlled by software.

The SSFE is implemented in C and is iteratively optimized with the feedback from compilation and profiling, ensuring highly efficient software-pipelining on the VLIW architecture. The final C code consists of around 400 lines, organized as (up to) 4-level loop-nests and 16 innermost loops.

2) Results: We summarize the results in Fig.II. With TMS320C6416 working at full frequency, the SSFE achieves 37.4 - 125.3 Mbps throughput for 4x4 64QAM transmission with $m = [1, 1, 1, 1]$, $[1, 1, 1, 2]$, $[1, 1, 1, 4]$ and $[1, 1, 2, 4]$. The SIMD feature is not exploited yet at this moment. With SIMD the throughput can be improved by a factor of 2. Note that the implementation on TMS320C6416 is just to investigate the architecture-friendliness of the SSFE with reproducible results. We are implementing the SSFE on a massive-parallel custom-platform with 16 FUs and 4-way SIMD datapath. In the ongoing implementation the throughput is expected to be between 100 to 250 Mbps for 4x4 64QAM transmissions. This is compatible with emerging wireless standards (100Mbps+).

TABLE II
SSFE ON TMS320C6416

SSFE with $\mathbf{m}$	$[1, 1, 1, 1]$	$[1, 1, 1, 2]$	$[1, 1, 1, 4]$	$[1, 1, 2, 4]$
Throughput (Mbps)	125.3	79.3	48.5	37.4

V. CONCLUSION AND FUTURE WORK

In this paper, we presented the SSFE as a novel near-ML detector specifically designed for parallel programmable baseband architectures. Comparing to the ASIC-minded K-Best, the SSFE not only significantly reduces the algorithmic complexity, but also results in a completely deterministic and regular dataflow in the algorithm. Hence, the SSFE can be easily parallelized and efficiently mapped on parallel programmable architectures. On a real life VLIW DSP platform, the SSFE brings nearly full efficiency of resource-utilizations.

At this moment, the SSFE works on only hard-output. In order to leverage advanced FEC (Forward Error Correction) techniques such as the Turbo codec and soft-input LDPC (Low Density Parity Check), we are now extending the SSFE with soft-output.

VI. ACKNOWLEDGEMENT

The authors thank Weiyu Xu (EE, Caltech, CA, USA) for his valuable comments on this work.

REFERENCES

[1] J, Rabaey, Keynote PresentationWireless Beyond the Third Generation - Facing the Energy Challenge *International Symposium on Low Power Electronic Design 2001*.

[2] S. Chen, T. Zhang, and Y. Xin, Relaxed K-best MIMO Signal Detector Design and VLSI Implementation, I*EEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, issue 3, pp. 328-337, March 2007

[3] D. G. et al., A 28.8 Mb/s 4 x 4 MIMO 3G CDMA receiver for frequency selective channels, *IEEE J. Solid-State Circuits*, vol. 40, no. 1, pp. 320C330, Jan. 2005. [4]

[4] D. G. et al.,, Silicon complexity for maximum likelihood MIMO detection using spherical decoding, *IEEE J. Solid-State Circuits*, vol. 39, no. 9, pp. 1544C1552, Sep. 2004.

[5] A. B. et al., VLSI implementation of MIMO detection using the sphere decoding algorithm, *IEEE J. Solid-State Circuits*, vol. 40, no. 7, pp. 1566C1577, Jul. 2005.

[6] Z. Guo and P. Nilsson, VLSI architecture of the Schnorr-Euchner decoder for MIMO systems, in *Proc. IEEE CAS Symp. Emerging Technol.*, 2004, pp. 65C68.

[7] Z. Guo and P. Nilsson, Algorithm and implementation of the k-best sphere decoding for MIMO detection, *IEEE J. Sel. Areas Commun.*, vol. 24, no. 3, pp. 491C503, Mar. 2006.

[8] K.-W. Wong, C.-Y. Tsui, R. S.-K. Cheng, and W.-H. Mow, A VLSI architecture of a K-best lattice decoding algorithm for MIMO channels, in *Proc. IEEE Int. Symp. Circuits Syst. 2002*, May 2002, pp. III-273CIII-276.

[9] A. Wiesel, X. Mestre, A. Pages, and J. R. Fonollosa, Efficient implementation on sphere demodulation, *in Proc. IEEE Workshop on Signal Processing Advances in Wireless Commun.*, Rome, Italy, Jun. 2003, pp. 36C40.

[10] R. Hoare, A. K. Jones, D. Kusic, J. Fazekas, J. Foster, S. Tung, and M. McCloud, Rapid VLIW processor customization for signal processing applications using combinational hardware functions, *EURASIP Journal on Applied Signal Processing*, vol. 2006.

[11] Y. Lin et al., "SODA: A Low-Power Architecture for Software Radio," *Proc. 33rd Ann. Intl Symp. Computer Architecture (ISCA 06),* IEEE CS Press, 2006, pp. 89-101.

[12] C. van Berkel et al., "Vector Processing as an Enabler for Software-Defined Radio in Handsets from 3G+WLAN Onwards," *Proc. 2004 Software Defined Radio Tech. Conf, SDR Forum*, 2004, p. B125.

[13] T. Kailath, H. Vikalo, and B. Hassibi, "MIMO Receive Algorithms," in Space-Time Wireless Systems: From Array Processing to MIMO Communications, Cambridge University Press, 2005.

[14] Nabar, R.U., Paulraj, A., Gore, D.A. and Bolcskei, H., An overview of MIMO communications-a key to gigabyte wireless. Proc. IEEE. v92. 198-218.

[15] L. G. Barbero and J. S. Thompson, "Rapid Prototyping of a Fixed-Throughput Sphere Decoder for MIMO Systems", in *IEEE International Conference on Communications (ICC '06)*

[16] D. Cescato, M. Borgmann, H. Bölcskei, J. Hansen, and A. Burg, "Interpolation-based QR decomposition in MIMO-OFDM systems," *Proc. of IEEE SPAWC*, 2005