Software Defined Radio Implementation of $K$-best List Sphere Detector Algorithm

Janne Janhunen, *Olli Silvén, Markku Juntti, Markus Myllylä
Centre for Wireless Communications University of Oulu, Finland
Olli Silvén, Janne Janhunen, Markku Juntti, Markus Myllylä)@ee.oulu.fi

Abstract—In this novel study, a real-valued signal model based on the $K$-best list sphere detector (LSD) algorithm is implemented to fixed-point digital signal processor (DSP). A $2 \times 2$ multiple-input multiple-output (MIMO) antenna system with 64-quadrature amplitude modulation (64-QAM) is assumed.

Our former studies proved that software sorting does not meet the real-time requirements, and, thus, in the current studies we assume a hardware sorter. The chosen list size $K=16$ is based on the simulation results carried out in MATLAB environment. We implemented the $K$-best LSD algorithm with Sandblaster multithreaded processor and achieved the throughput of 17.9 Mbps, when the hardware sorter was assumed beside the digital signal processor.

This novel study shows that the general-purpose digital signal processor has potential to achieve high throughput, when hardware accelerated sorter is assumed. In the current study, the latency of the control code and partial Euclidean distance (PED) calculations were decreased, but the latency of memory loads and stores are significant. We will also compare results from x86 processor architecture and application-specific instruction set processor (ASIP) implemented by using transport triggered architecture (TTA), in which the same parameters were used. The TTA has benefits compared to DSPs, especially in data transmission.

I. INTRODUCTION

In the third generation partnership project (3GPP) long term evolution (LTE) targets, 100 Mbps is planned to be transmitted through wireless connections [1]. High-rate wireless communication needs power efficient solutions to process the increasing amounts of data with limited hardware and low power consumption. The multiple-input multiple-output (MIMO) antenna system combined with the orthogonal frequency division multiplexing technique (MIMO–OFDM) has been included in multiple wireless telecommunication standards, such as the IEEE 802.11 WLAN, IEEE 802.16 wireless metropolitan area network (WMAN), Worldwide Interoperability for Microwave Access (WiMAX) and the 3GPP LTE. The multipath environment causes MIMO channel to be frequency-selective and OFDM can transform such a channel into a set of parallel frequency-flat MIMO channels. The transform into a frequency-flat MIMO channel decreases the receiver complexity.

Because multiple standards are being proposed for the wireless and wired communication, flexibility is required from the terminal device. The advantage of software defined radio (SDR) is its flexibility in an environment with multiple standards. However, the increased computing loads of the future applications cause new challenges to the SDR implementations.

The maximum likelihood (ML) detector is optimal for finding the closest lattice point [2]. However, it is not often feasible for real implementations, because its computational complexity increases exponentially with the increasing number of transmit antennas. The sphere detector (SD) [3] calculates the ML solution with reduced complexity compared to full-complexity exhaustive search ML detectors [2]. The list sphere detector (LSD) [4] is a variant of the sphere detector that can be used to approximate the soft decision maximum a posteriori probability (MAP) detector.

We studied the performance of the multithreading Sandblaster digital signal processor (DSP) [5] to fulfill the real-time requirements of the $K$-best LSD algorithm. The $K$-best LSD algorithm [6] is a breadth-first search algorithm based on the well known $M$-algorithm [7], [8]. The $K$-best algorithm proceeds one level at the time by calculating the partial Euclidean distances (PED) of all the admissible nodes and sorting the $K$ best candidates for the next level. The main complexity of the $K$-best LSD algorithm is the calculation of the PEDs and sorting the $K$ best distances into the list. The algorithm is serial between the PED calculation and sorting, which prevents writing a fully parallel code between the levels. On the other hand, the throughput and computational complexity can be fixed, and, thus, a parallel and pipelined implementation is possible inside the algorithm. High computing power is required to achieve the real-time requirements.

The rest of the paper is organized as follows. The system model and maximum likelihood detection are briefly presented in Section II. The list sphere detection algorithm, $K$-best algorithm and simulation results are presented in Section III. Sandblaster processor, implementation and results have been presented in Sections IV–VI, respectively. Section VII compares Sandblaster, x86 and transport triggered architecture (TTA) implementations. Section VIII presents two illustrating examples of subcarrier allocation in MIMO–OFDM system. The final section concludes the paper.
II. System Model

A MIMO–OFDM based multiple antenna system is assumed with \( N \) transmit and \( M \) receive antennas. Figure 1 presents the block diagram of the MIMO–OFDM transmission architecture. In this study, the list sphere detector block of the receiver is implemented to fixed-point DSP.

The received signal on \( s \)th subcarrier can be presented as

\[
y_s = H_s x_s + n_s, \quad s = 1, 2, ..., S
\]

where \( S \) is the number of subcarriers, \( y_s \in \mathbb{C}^{M \times 1} \) is the received signal vector, \( x_s \in \mathbb{C}^{N \times 1} \) is the transmitted symbol vector and \( n_s \in \mathbb{C}^{M \times 1} \) is the noise vector. The symbol \( H_s \in \mathbb{C}^{M \times N} \) denotes the channel matrix. The entries of \( x_s \) are chosen independently of each other from a quadrature amplitude modulation (QAM) constellation.

The ML detector minimizes the Euclidean distance between the received signal \( y \) and the lattice points \( Hx \) and selects the lattice point that minimizes the Euclidean distance to the received vector \( y \), i.e.,

\[
\hat{x} = \arg \min_{x \in A^N} \| y - Hx \|^2,
\]

where \( A \) is the symbol alphabet and \( \| \cdot \| \) denotes the Frobenius norm of a vector. The exhaustive search can be used to solve the ML detection problem. However, it becomes computationally infeasible as the set of lattice points increases. The sphere detection algorithm solves the ML approximation (2) by limiting the search to the lattice points that lie inside a \( M \)-dimensional hyper-sphere [2]. Figure 2 illustrates a single-input single-output (SISO) antenna system with 64-QAM. On the left side, the original constellation is presented with the white circles and the black circle is the transmitted symbol. On the right side, the grey circle illustrates the received symbol, which is placed somewhere between the lattice points due to additive noise in the channel. At this point, the maximum likelihood method would calculate the Euclidean distance between the received symbol and every constellation point, whereas the SD calculates the distances to constellation points inside the circle of a certain radius, and, thus, reduces the computational complexity significantly. The dimension of the circle depends on the number of receiver antennas.

Fig. 1. Block diagram of the MIMO–OFDM transmission.

III. List Sphere Detector

The LSD algorithm is used to approximate the MAP detection in channel coded systems with reduced computational complexity. Basically, the LSD algorithm traverses a tree, whose depth depends on the number of transmit antennas and the number of branches depends on the used constellation. The real signal model doubles the depth in the search tree compared to complex signal model algorithm, but provides also a less complex distance calculation.

The computational complexity can be reduced by limiting the search to inside a sphere with radius \( d \) using the sphere constraint \( d^2 \geq \| y - Hx \|^2 \). The channel matrix \( H \) can be QR decomposed (QRD) into two parts. If the number of transmit and receiver antennas are equal, the channel matrix can be presented as \( H = QR \), where \( Q \) denotes a \( N \times N \) orthogonal matrix and \( R \) is a \( N \times N \) upper triangular matrix. After the QR decomposition, the equation can be rewritten as

\[
d^2 \geq \| y - QRx \|^2
\]

where \( Q^H \) denotes the Hermitian transpose of matrix \( Q \). By denoting \( Q^H y = y' \), we get

\[
d^2 \geq \| y' - Rx \|^2.
\]

Let \( x_i = (x_i, x_i+1, ..., x_{i-1}, x_N)^T \) denote the last \( N-i+1 \) components of the vector \( x \). The sphere search can be thought as a tree structure, where the root layer corresponds to \( x_i^N \). The last elements of the possible symbol vectors are calculated first, i.e., \( x_N \) and then \( x_{N-1} \) and so on.

The partial Euclidean distance can be calculated as [9], [10]

\[
d(x_i^N) = d(x_{i+1}^N) + \left| y_i' - \sum_{j=1}^{N} r_{i,j}x_j \right|^2,
\]

where \( i = N, N-1, ..., 1 \) and \( r_{i,j} \) is the \( i,j \)th term of the upper triangular matrix \( R \).

A. K-best List Sphere Detector

The \( K \)-best LSD algorithm provides a fixed throughput and complexity, and, thus, it is an interesting alternative for implementation. The complexity of the algorithm depends mostly on the number of transmit antennas, the list size and the modulation method. The algorithm maintains a list of the \( K \) best symbol candidates and the corresponding multidimensional constellation symbol identifier. For example in 64-QAM

Fig. 2. In sphere detection, the Euclidean distance calculations are limited to inside a sphere.
with a real-valued signal model, the $\sqrt{64} = 8$ constellation points can be represented with $S_0 = \log_2(8) = 3$ bits, 000 representing the first constellation point and 111 representing the last constellation point. By setting the sphere radius to infinity, $d = \infty$, a fixed number of nodes are processed in the $K$-best algorithm. The selection of the infinite sphere radius is based on the basic idea of the sphere detector that only the lattice points inside a hyper-sphere with a certain radius are taken into account in the distance metric minimization (2).

The $K$-best LSD algorithm is based on the breadth-first strategy, i.e., the algorithm proceeds one level at the time by calculating all the admissible PEDs and storing the $K$ best PEDs to the intermediate list in memory. The search will continue with all admissible nodes on the next level until the leaf nodes are reached. After the final level, the $K$ best candidates are sorted and output as a final candidate list. However, it should be noted that the final candidate list may not include the lowest EDs.

Maintaining the candidate list requires continuous memory usage, and, thus, it is the most critical part of the algorithm. A large list size improves the BER performance but leads to increasing computation burden and memory usage. The candidate list updating requires a comparison between a new PED and the maximum PED in the list. If the new PED is smaller than the maximum PED in the list, the new PED is sorted into the list. Otherwise, the list stays untouched. The $K$-best LSD algorithm is described as Algorithm 1 [6].

Figure 3 presents the search tree pruning in the $K$-best algorithm, where the list size $K = 2$. A real signal model, $2 \times 2$ antenna system and 4-QAM or $4 \times 4$ antenna system and binary phase-shift keying (BPSK) modulation is illustrated. The black circles show the $K$ best nodes at each level and the grey circles are the pruned nodes, which did not succeed in the selection. Nodes that have not been visited at all, are shown as white circles. It should be noted that the impact of tree pruning becomes more significant when the number of transmit antennas increases, a higher order modulation is used and the list size is small.

**Algorithm 1 (K-best LSD algorithm):**

Preprocessing:
- QR decomposition of the channel matrix $\mathbf{H}$.
- Inputs: $\mathbf{R}$, $\mathbf{y} = \mathbf{Q}^T \mathbf{y}$, $K$

Algorithm:
1. Start with empty candidate set from the root layer.
2. Denote the partial candidate set by $\mathbf{x}_{i+1}$.
3. Sort the partial candidates according to their PEDs and store the $K$ lowest PEDs.
4. If the last level is calculated (candidates are leaf nodes), stop the algorithm and give the candidates and their EDs as outputs. Otherwise, continue to step 2 with the stored candidates.

### B. Simulation Results

A good compromise of the list size and word lengths was decided by running simulations. The list size has a significant impact on the complexity of the LSD algorithm. Parameter studies were performed on a MIMO–OFDM simulator running in MATLAB environment. In the simulator, one frame corresponds to one OFDM symbol and consists of 300 individual symbol vectors, each mapped to one OFDM subcarrier. The simulations were run with 1000 frames per each SNR point. Table I presents the simulation parameters inspired by the 3G LTE specifications [11], [12], [13].

The simulator takes into account the effect of log-likelihood ratio (LLR) clipping [14] with threshold $L_{\text{max}} = 8$. The LSD output list is used to calculate the approximation of the probability LLR of each transmitted bit. By limiting the dynamic range of the LLR, the required LSD list size can be decreased and the computational complexity of the LSD decreases.

The corresponding bit error rate (BER) curve is presented in Figure 4. Based on the BER curve, the list size of 16 was found out to be a good compromise between complexity and performance for the implementation. Respectively, a 16-bit fixed-point arithmetic (6-bit integer, 10-bit fraction) was chosen, because significant decrease in performance was not noticed when compared with the floating point performance.

### IV. Sandblaster Processor

The Sandblaster processor responds to the new challenges of a mobile communication and multimedia requirements in hand-held devices. The processor features include compound instructions, single input multiple data (SIMD) vectorization units and a hardware support for multiple threads. The Sandblaster 3011 architecture has four cores each having eight
In the Sandblaster multithreaded processor, all the hardware threads can operate simultaneously, and, thus, multiple concurrent program execution is supported. This feature has a great advantage in implementations, where parallelism is required, e.g., processing multiple parallel subcarriers in the MIMO–OFDM receiver implementation. The multithreaded processor uses a so-called Token Triggered Threading (T3), which simplifies the thread selection hardware, and, thus, leads to the power savings compared to the conventional simultaneous multithreading [15]. Flexibility in scheduling threads, SIMD vector operations, and compound instructions provides higher performance than conventional interleaved multithreading. All threads can simultaneously execute instructions, but only one thread may issue an instruction on a cycle boundary [16].

Multithreading covers the latency of long pipeline instructions. On the other hand, long pipelines allow the use of single port register files. Because the write back stages are staggered, the functionality of the multiple write ports is provided even though a single port register file is implemented. The token triggered threading enables the use of banked register files [16]. This makes it possible to use one register file bank, while an odd thread accesses the alternate register file bank. The same characteristics enable the use of banked single ported level-1 (L1) memories. The Sandblaster implementation of the multithreading allows the processor cycle time to be decoupled from the memory access time, and, thus, logic and memory can operate in the linear region of the power consumption.

In the Sandblaster processor architecture, most SIMD vector instructions have eight pipeline stages. Since there are eight processor cycles between instructions executed by the same thread, the latency is hidden to a single thread cycle. Dependency check or bypass hardware is not required, because result of the instruction is guaranteed to be written back before the same thread issues a new instruction. Figure 6 illustrates the instruction execution on the Sandblaster processor and conventional DSP. Figure 6(a) represents the instruction execution from the single thread point of view in a multithreaded processor. Figures 6(b) and 6(c) illustrate the instruction execution from the time usage point of view. In the Sandblaster processor, most of the vector instructions take advantage of the eight processor cycles. It means that each thread has eight processor cycles to execute the instruction before it can issue the next one. Figure 6(c) presents the typical case in a conventional signal processor, where NOP (no operation) cycles are used to keep computation valid. The high number of NOPs is usually a consequence of the high latency operations, such as load operation. The simplified examples in Figure 6, show the gain of the multithreaded processor, where eight hardware threads can execute 16 instructions in the same time as the conventional processor executes five instructions.

Multithreading covers the latency of long pipeline instructions. On the other hand, long pipelines allow the use of single port register files. Because the write back stages are staggered, the functionality of the multiple write ports is provided even though a single port register file is implemented. The token triggered threading enables the use of banked register files [16]. This makes it possible to use one register file bank, while an odd thread accesses the alternate register file bank. The same characteristics enable the use of banked single ported level-1 (L1) memories. The Sandblaster implementation of the multithreading allows the processor cycle time to be decoupled from the memory access time, and, thus, logic and memory can operate in the linear region of the power consumption.

Table II and Figure 5 present the processor properties and architecture, respectively.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of processors</td>
<td>4</td>
</tr>
<tr>
<td>Number of hardware threads</td>
<td>32</td>
</tr>
<tr>
<td>Gate count (kGE)</td>
<td>4×300</td>
</tr>
<tr>
<td>Processor clock rate (MHz)</td>
<td>600–800</td>
</tr>
<tr>
<td>Thread clock rate (MHz)</td>
<td>75</td>
</tr>
<tr>
<td>MIPS</td>
<td>9600</td>
</tr>
<tr>
<td>Power (mW)</td>
<td>4×125</td>
</tr>
<tr>
<td>mW/MIPS</td>
<td>0.05</td>
</tr>
<tr>
<td>Supply (V)</td>
<td>0.9</td>
</tr>
</tbody>
</table>

![Fig. 5. Sandblaster processor architecture.](image-url)
Fig. 6. Instruction execution presented (a) from the thread point of view. (b) Instruction execution on Sandblaster processor and (c) on conventional DSP from the time point of view. NOP stands for no operation clock cycle.

hardware sorter with the latency of a single thread cycle could be practical to implement without violating the database of the Sandblaster architecture. Several sorter solutions for the K-best LSD algorithm have been studied in [19], [20]. A single cycle insertion sorter, used in [19], is a reasonable alternative, because its simple implementation. However, the fact that the Sandblaster processor can hide eight processor cycles makes a variable latency heap sorter also an interesting choice. In this study, an efficient hardware sorter design for the Sandblaster processor is not included.

The serial nature of the K-best algorithm limits the parallel thread usage inside the symbol vector calculation. Thus, it is sensible to pin each symbol vector (subcarrier) calculation to a certain hardware thread. This guarantees the most predictable scheduling with the lowest synchronization overhead and reduces the mutual exclusion synchronization. On the other hand, if all the hardware threads are pinned, no unpinned software threads can run until one of the pinned hardware thread exits and releases the resources. The Sandblaster processor support also other scheduling methods, but because of the strict latency requirements and the straightforward implementation of the K-best algorithm, other scheduling methods were not considered.

There are many coding techniques for an efficient code generation presented in literature. When programming the Sandblaster processor, it is important that the code supports vectorization. The vectorization is a multi-step process, in which the compiler analyzes the loops and then performs the vectorization. If there is a possibility that the code will break, the compiler will not vectorize the loop. There are multiple steps to ensure an efficient vectorization and the algorithm has to be modified partly based on the following restrictions. Memory accesses in the loop have to be independent of each other so that the semantics of the code do not change due to the read-write reordering in the vectorization. To guarantee maximum benefit from the vectorization unit, the length of the loops should be divisible by four and the data type should be short (16 bits), because each vector register can hold four 16-bit data units. The list size was set to \( K = 16 \) and in 64-QAM with a real-valued signal model there are \( \sqrt{64} = 8 \) constellation points, which have a straightforward impact on an efficient vectorization. The short data type (16 bits) is used to store the PED and the corresponding symbol identifier. Vectorization is disabled if conditional statements are inside the loop. Our implementation includes four search levels due to two transmit antennas, 64-QAM and real-valued signal model. A single C function was implemented for each level. This approach was found out to be good, because we managed to vectorize the whole code, and, hence, the processor performance was exploited efficiently.

The computation of the algorithm proceeds as presented in Algorithm 1. The 3rd level calculates PEDs for eight different values of \( x_3 \) (constellation points) as

\[
D_3 = \left| y'_3 - \sum_{j=3}^{3} r_{3,j} x_j \right|^2 = \left| y'_3 - r_{3,3} x_3 \right|^2, \quad (6)
\]

where the first value of \( x_3 \) can be denoted as \((x_3)_0\) and the last with \((x_3)_7\). The 3rd level can be implemented in a single for-loop as presented in Figure 7, where the value of \( x_3 \) changes on every iteration. The sorting is not required on the third level, because the constellation size \( \sqrt{64} = 8 < K = 16 \). Thus, the order of array \( x_3 \) remains the same for the 2nd level. The pseudocode illustrates that the PED calculation can be executed by using load, store, add, multiplication and shift operations. On levels 2, 1 and 0 the code is more complex, because the PEDs and the corresponding symbol identifiers from the previous levels are included in calculation. For example, the PEDs of the last level (0th level) are computed as

\[
D_0 = D_1 + D_2 + \left| y'_0 - r_{0,0} x_0 + r_{0,1} x_1 + r_{0,2} x_2 + r_{0,3} x_3 \right|^2. \quad (7)
\]

The calculation of (7) can be separated to four vectorized for-loops to decrease the computational complexity and latency. The first loop executes \( r_{0,1} x_1 + r_{0,2} x_2 + r_{0,3} x_3 \) calculation and stores the intermediate result. The second loop calculates \( y'_0 - r_{0,0} x_0 \). The third loop loads the stored PEDs \((D_1, D_2)\) and the corresponding symbol identifiers. The fourth loop, which is the inner loop of the third loop, calculates the new PEDs and updates the corresponding symbol identifiers.

for(i = 0; i < constellation size; i++)
{
    accu = y'_i;
    accu -= ((r_{3,3} * (x_3)_i) >> FRACTION);
    PED_memory[i] = (accu * accu) >> FRACTION;
}

Fig. 7. Pseudocode (C-style) presentation of the 3rd level PED calculations.

A register based implementation is not possible, because storing the PEDs and the related constellation symbol identi-
fiers would require 32 registers. Thus, intermediate results are stored into the memory. Short data type is used, because the Sandblaster processor has a 64-bit data path to memory, i.e., the processor can store/load four 16-bit quantities efficiently. Small memory stacks are used to store constellation symbol identifiers and sorted distance list. The core specific level-1 memory is used to keep the memory latencies as low as possible.

### VI. Results

A single symbol vector requires 328 PED calculations. Table III presents the latencies of the distance calculations in all four levels with and without memory latencies, respectively. The third level is the simplest. Eight distances are calculated in 40 thread cycles according to the constellation symbols. The second level calculates 64 distances in 129 thread cycles. The 16 shortest distances are sorted to the intermediate list. When proceeding toward the next levels, the distance calculation complexity increases due to more complex symbol fetching from the symbol list. On the first and zeroth levels, 128 PEDs are calculated in 243 and 262 thread cycles, respectively.

The decoding throughput for the implementation is calculated as

\[ T_D = \frac{N \times S_b}{L_{cc}} \times f, \tag{8} \]

where \( L_{cc} \) is the latency of a output symbol vector and \( f \) is the operating frequency.

Totally, 328 PEDs are calculated in 674 thread cycles. The PED calculation with a single hardware thread, hence, reaches a performance of 1.33 Mbps. If all the 32 hardware threads of the Sandblaster processor are allocated for the PED calculation, performance of 42.7 Mbps is achieved. The simulation, where the memory latencies are included\(^1\), achieves a detection throughput of 17.9 Mbps, when all the hardware threads are used. Comparing the total latencies in Table III, shows that the impact of memory latencies in the \( K \)-best implementation is significant and decreases the performance substantially.

<table>
<thead>
<tr>
<th>Level</th>
<th># of PEDs</th>
<th># of Cycles (with memory)</th>
<th># of Cycles (without memory)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3rd</td>
<td>8</td>
<td>209</td>
<td>40</td>
</tr>
<tr>
<td>2nd</td>
<td>64</td>
<td>306</td>
<td>129</td>
</tr>
<tr>
<td>1st</td>
<td>128</td>
<td>532</td>
<td>243</td>
</tr>
<tr>
<td>0th</td>
<td>128</td>
<td>561</td>
<td>262</td>
</tr>
<tr>
<td>tot.</td>
<td>328</td>
<td>1608</td>
<td>674</td>
</tr>
</tbody>
</table>

Above, the significance of the memory latencies to the implementation was pointed out. The core specific level-1 memory was used, but to reduce latencies further, memory stores need to be reduced on the algorithm level. A straightforward solution to reduce memory latencies is to decrease the list size \( K \). In order to decrease the list size without decreasing the BER performance, an efficient preprocessing method is required and this is clearly one of the most interesting research topic in the future research.

The implementation includes extra memory usage to store the intermediate distance values after they are calculated on each level. The throughput of 42.7 Mbps can be thought as a upper bound of the processor performance. Respectively, the throughput of 17.9 Mbps can be thought as a lower bound of the processor performance.

Table IV presents the memory usage of the implementation provided by the Sandblaster development tool. The data is allocated to the L1 memory. A single Sandblaster processor has eight 8k bytes banks of L1 data memory, and, thus, the size of the memory is not an issue for the detector implementation. As can be seen, the percent of the cache hits is high, which proofs that the instruction cache is also used efficiently.

<table>
<thead>
<tr>
<th></th>
<th>Memory Usage in Sandblaster Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction memory</td>
<td>3224 bytes (8-bit)</td>
</tr>
<tr>
<td>Data memory (L1)</td>
<td>888 bytes (8-bit)</td>
</tr>
<tr>
<td>Cache hits</td>
<td>99.866%</td>
</tr>
</tbody>
</table>

### VII. Implementation Comparison

In general, the comparison between implementations is difficult for several reasons. First of all, parameters, such as the number of antennas, the modulation method, the word length, the list size and the system model should be fixed. In addition, implementations on different platforms, such as application specific integrated circuits (ASICs), application-specific instruction set processors (ASIPs), DSPs, field-programmable gate arrays (FPGAs) and x86 processor architecture, make the comparison even more difficult. Recently at CWIC, we have studied the \( K \)-best LSD algorithm on DSP, ASIP and x86 platforms. We have fixed the parameters, and, thus, the achieved latency and throughput results are comparable even though the platforms are different.

The previous study with the Texas Instruments TMS320C6711 very long instruction word (VLIW) processor [17] was the starting point. It is not directly comparable with the current Sandblaster DSP study, because a \( 4 \times 4 \) antenna system, 16-QAM, list size \( K = 63 \) and complex signal model was previously assumed. However, we learned that the software sorting is not fast enough for the \( K \)-best algorithm and now we assume a hardware sorter beside the digital signal processor. We also improved the coding technique and found out several new strength reduction methods to speed up the implementation.

The increasing trend of using general-purpose processors in hand-held devices motivated us to experiment the \( K \)-best LSD algorithm on the x86 processor architecture. The results were promising considering that the processor architecture is designed for general purpose processing. We used the Microsoft Visual C++ compiler and profiled the latencies by

\(^1\)This result includes an extra memory usage, because sorter was not included into the simulator, and, thus, the intermediate distances were stored into the memory after the distance calculations.
using the time stamp counter (TSC) provided by the Intel IA-32 instruction set. With the TSC high-resolution timer, cycle accurate latency is reached. Processing a single symbol vector on the x86 architecture spend approximately 1674 clock cycles without the sorter latency. The level-1 memory latencies were included in the results. The performance is close to the results achieved with the Sandblaster processor, because both platforms have similarities to VLIW type of architecture. However, it should be noted that the power consumption and the capability of hiding the sorter latencies are much better on the Sandblaster device. This brief glance into the x86 processor architecture reminds us about the trend we are heading to in hand-held devices, but also pointed out the drawbacks of the general-purpose processors.

The ASIP implementation based on the TTA [21] was carried out in parallel with the Sandblaster DSP implementation. The same parameters were chosen to guarantee the latency and throughput comparison between the implementations. The implementation platforms are totally different, which complicates the comparison, but on the other hand it provides a novel comparison arrangement.

TTA is a programmable architecture, where processor is a built of independent function units and register files, which are connected with transport buses and sockets [22]. In conventional processor architectures, the data transmission may become the bottle-neck of implementations, where high amount of data is processed. In the transport triggered architecture, the obstacle is solved by reversing the operations to be consequences of data transports. In TTA design, the processor designer has free hands to build the optimal data transmission by adding enough buses between logic and memory.

The current TTA implementation [21] achieves the throughput of 7.6 Mbps with hardware complexity of 25 kGE (gate equivalents) and 280 MHz processor clock frequency. A single symbol vector is processed in 441 clock cycles. In addition to the special function unit sorter, the implementation includes general-purpose function units, which makes it more interesting compared to the previous TTA implementation [23], [24]. Despite the fact that the processor was designed specially for the K-best algorithm, the function units, such as adder, subtracter and multiplier provides the processor usage with limitations for other purposes too. The processor include no extra functions, and, thus, it can not be considered as a general-purpose processor. However, without the extra function units and buses, the processor implementation is rather simple and the gate count is low.

If we compare a DSP and a transport triggered architecture, the greatest benefit of the TTA solution is the possibility to implement an optimal data transmission bus. Arithmetic operations are executed efficiently also with DSPs, but the data transmission between memory and function units is the bottle-neck in highly parallel algorithms.

VIII. SUBCARRIER ALLOCATION IN MIMO–OFDM SYSTEM

Let us consider the subcarrier allocation in a very straightforward way to clarify the real-time constraints in MIMO–OFDM system. Inspired by the 3GPP LTE specifications (Table V), a single OFDM symbol consists of 512 (300 used) individual symbol vectors, each mapped to one OFDM subcarrier. Seven OFDM symbols should be processed in 0.5 ms, and, thus, symbol duration of 71.42 µs is required. In other words, 300 symbol vectors should be processed in 71.42 µs. If we assume the Sandblaster 3011 device (Table II), which has 32 hardware threads each running on 75 MHz clock frequency, we can roughly calculate the real-time requirements for the list sphere detector. 32 symbol vectors can be processed in parallel and ten sequential calculations are required, which means that a single thread can spend approximately 7.14 µs to calculate one symbol vector. With 75 MHz clock frequency, this means 536 thread cycles. Comparing the calculated result to the results we have presented in Section VI, the real-time requirement is not achieved with a single Sandblaster 3011 device, but at least two parallel processing devices are required.

<table>
<thead>
<tr>
<th>Number of subcarriers</th>
<th>512 of which 300 used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth</td>
<td>5 MHz</td>
</tr>
<tr>
<td>Sub-frame duration</td>
<td>0.5 ms</td>
</tr>
<tr>
<td>Carrier frequency</td>
<td>2.4 GHz</td>
</tr>
<tr>
<td># of OFDM symbols per sub-frame</td>
<td>7/6</td>
</tr>
<tr>
<td>Cyclic prefix (CP) duration</td>
<td>4.69 µs</td>
</tr>
<tr>
<td>Symbol duration</td>
<td>71.42 µs</td>
</tr>
</tbody>
</table>

Let us calculate another real-time requirements example. A single TTA processor has a clock frequency of 280 MHz, and, thus, one clock cycle duration is \( \frac{1}{280 \text{MHz}} \approx 3.57 \text{ns} \). Now, let us calculate how many clock cycles can be allocated for a single OFDM symbol to achieve the real-time requirements. As we already pointed out, there are 71.42 µs to process 300 symbol vectors and a clock cycle with 280 MHz processor takes 3.57 ns. According to this assumptions, there are 19,997 clock cycles time to process 300 symbol vectors, which means that each symbol vector has to be processed in 66 clock cycles. As presented in Section VII, the TTA processor executes a single symbol vector in 441 clock cycles, and, thus, seven TTA processors are required for the real-time processing.

The purpose of the examples was point out that not only the throughput is important, but also the strict real-time requirements need to be considered in the detector design. The examples also show that parallel processing of the symbol vectors are required to meet the real-time constraints. It is obvious that programmable processors used in the current mobile devices have difficulties in achieving the required performance. Table VI sums up the hardware requirements to achieve the real-time constraints in the MIMO–OFDM system.
TABLE VI
HARDWARE REQUIREMENTS FOR THE REAL-TIME PROCESSING

<table>
<thead>
<tr>
<th></th>
<th>Sandblaster 3011</th>
<th>TTA processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of processors</td>
<td>2 devices (8 processors)</td>
<td>n/a</td>
</tr>
<tr>
<td>Gate count (KGE)</td>
<td>2,400</td>
<td>175</td>
</tr>
<tr>
<td>Power consumption (mW)</td>
<td>8×125</td>
<td>n/a</td>
</tr>
<tr>
<td>Throughput (Mbps)</td>
<td>35.8–85.4</td>
<td>53.2</td>
</tr>
</tbody>
</table>

IX. CONCLUSIONS

We studied the performance of the multithreading Sandblaster processor for the K-best LSD algorithm implementation. In the current studies, we achieved already tens of Mbps throughput with the general-purpose Sandblaster digital signal processor, when the hardware sorter was assumed. However, the throughput of 100 Mbps, presented in 3GPP LTE requirements [1], is still beyond the current implementation. We took significant steps since previous studies [17] and we will continue our studies by implementing other LSD algorithms with programmable processors to find out the hardware requirements and the power consumption estimates of the future mobile devices.

We compared the performance and benefits of the DSP and TTA processor implementations. These two implementations were comparable, because the same parameters were used. The benefit of the TTA compared to DSP was pointed out to be the possibility to design fast enough data transmission for the algorithm. The brief glance into the performance of the x86 processor architecture reminded us about the general-purpose processor trend we are possibly heading to in handheld devices. We also pointed out with two examples that the detector design requires parallel symbol vector processing to meet the real-time requirements according to the 3GPP LTE specifications.

X. ACKNOWLEDGMENTS

Numerous people have directly and indirectly contributed to this paper. Special thanks to J. Parviainen from Sandbridge Technologies helping with Sandblaster processor related issues as well as J. Antikainen from CWC, providing us the TTA implementation results and J. Ketola from CWC, helping with the simulator related issues.

REFERENCES