Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2007, Article ID 54173, 14 pages
doi:10.1155/2007/54173

Research Article

Application-Specific Instruction Set Processor Implementation of List Sphere Detector

Juho Antikainen,[1] Perttu Salmela,[2] Olli Silvén,[1] Markku Juntti,[1] Jarmo Takala,[2] and Markus Myllylä[1]

[1] *Information Processing Laboratory and Centre for Wireless Communications, University of Oulu, 90014 Oulu, Finland*
[2] *Institute of Digital and Computer Systems, Tampere University of Technology, 33101 Tampere, Finland*

Received 8 June 2007; Revised 18 October 2007; Accepted 12 November 2007

Recommended by Marco Platzner

Multiple-input multiple-output (MIMO) technology enables higher transmission capacity without additional frequency spectrum and is becoming a part of many wireless system standards. Sphere detection has been introduced in MIMO systems to achieve maximum likelihood (ML) or near-ML estimation with reduced complexity. This paper reviews related work on sphere detector implementations and presents an application-specific instruction set processor (ASIP) implementation of K-best list sphere detector (LSD) using transport triggered architecture (TTA). The implementation is based on using memory and heap data structure for symbol vector sorting. The design space is explored by presenting several variations of the implementation and comparing them with each other in terms of their latencies and hardware complexities. An early proposal for a parallelized architecture with a decoding throughput of approximately 5.3 Mbps is presented

Copyright © 2007 Juho Antikainen et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. INTRODUCTION

Multiple-input multiple-output (MIMO) communications based on multiple transmit and receive antennas will be applied in several wireless communication system standards to increase the spectral efficiency and the data rates. Timely examples include the evolving third generation (3G) cellular systems known as 3G long term evolution (LTE) and worldwide interoperability for microwave access (WiMAX) system. Multiple antennas can in general be utilized to implement either spatial transmit or receive diversity, beamforming in smart antennas or spatial multiplexing (SM) sometimes called layering of multiple data streams. This poses remarkable challenges for the MIMO detector and receiver baseband design.

The theoretical capacity potential of MIMO communications has been analyzed in [1–3]. A practical SM scheme called Bell Laboratories Layered Space-Time (BLAST) architecture [3, 4] has been proposed and shown to be able to realize the theoretically predicted capacity gains at least to some extent. For a more complete overview on the rich literature on MIMO communications, see, for example, [5–7] and references therein.

Transmission of independent data streams from different antennas in SM-MIMO systems usually causes spatial multiplexing interference (SMI) or interantenna interference. This calls for sophisticated receiver designs to cope with the interference. The optimal detector would be the maximum a posteriori (MAP) symbol detector providing soft outputs or log-likelihood ratio (LLR) values to the forward error control (FEC) decoder. Since the computational complexity of both MAP and ML sequence detectors depends exponentially on the number of spatial channels, several suboptimal solutions have been proposed and studied.

Linear minimum mean square error (LMMSE) or zero forcing (ZF) detection principles can be straightforwardly applied in MIMO detection [8]. However, the linear detectors can suffer a significant performance loss in fading channels, in particular with spatial correlation between the antenna elements [9]. Ordered serial interference canceller (OSIC) was proposed already in the original papers considering the BLAST architecture [3, 4, 10].

An ML detector approximation based on the sphere detector [11] for MIMO communications has been introduced in [12]. Another research line has considered the concept of lattice reduction to the MIMO detector problems [13–16]. Other important detector techniques include the list sphere detector (LSD) [17], iterative tree search detection schemes [18] and layered structure maximum likelihood detection scheme [19]. The sphere detectors are particularly

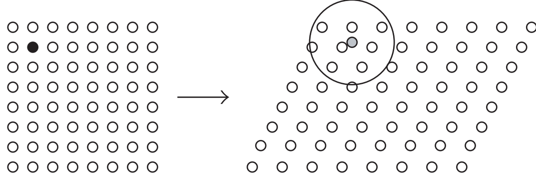


Figure 1: The basic idea of sphere detection.

interesting, since their expected and worst case complexities have been found to be only polynomial and often cubic in practically relevant signal-to-noise ratio (SNR) regimes [20, 21]. Their practical feasibility is further supported recently by practical implementations reported in the literature [22–26]. Therefore, we focus on the LSD algorithms in our treatment herein.

The LSD algorithm has several variants, see, for example, [26] for a more complete discussion. In practical implementations, so called K-best list sphere detector [27] (K-best LSD) has received significant attention. It belongs to the general class of breadth-first trellis search algorithms [28] and is actually a variant of the well known M algorithm [29, 30]. It has numerous good implementation properties, like constant throughput and pre-determined complexity.

Application-specific integrated circuits (ASICs) have been conventionally used for tasks that demand high computational resources and low power consumption. However, their design can be very laborious and software-based algorithm modifications are often very limited. General purpose digital signal processor (DSP) solutions can typically provide the flexibility, but do not often provide enough computation power to satisfy the stringent requirements of high speed real time communications with terminal level power consumption constraints. Application-specific instruction set processor (ASIP) solutions can provide a possibility to reduce design and production costs and still enable meeting the high performance requirements of MIMO receiver algorithms.

In this paper, we design an application-specific instruction set processor for K-best list sphere detector using the TTA [31–33] computation paradigm. The work is based on our previous conference publications [34, 35], compared to which this paper presents a wider and more detailed presentation and includes a review of related work. Inspired by the evolving 3G cellular systems [36], we use a 4×4 MIMO system with 16-quadrature amplitude modulation (16-QAM) as our base line design target. We operate directly on the complex-valued constellation points. As the design work is done with real-life implementation in mind, fixed point arithmetic is used. The goal is to try to achieve low energy consumption, so using memory is preferred extensively over registers. The design space is investigated by comparing the latency and complexity of the implementation to several proposed variations of it.

The paper begins by defining the MIMO communication problem and the appropriate receiver algorithms in Section 2. Related work on sphere detector implementations is discussed in Section 3. Section 4 describes the ASIP LSD implementation in detail along with several variations that could be used for improving the decoding throughput. The latency and hardware complexity of the implementation and different alternatives are estimated and compared in Section 5, and conclusions are drawn in Section 6.

2. MIMO RECEIVER ALGORITHMS

A MIMO communication system with M receive and N transmit antennas can be modeled using the equation

$$\mathbf{x} = \mathbf{H}\mathbf{s} + \mathbf{n}, \tag{1}$$

where $\mathbf{x} \in \mathbb{C}^{M\times 1}$ is the vector of received symbols, $\mathbf{H} \in \mathbb{C}^{M\times N}$ is the channel matrix, $\mathbf{s} \in \mathbb{C}^{N\times 1}$ is the vector of transmitted symbols, and $\mathbf{n} \in \mathbb{C}^{M\times 1}$ is the Gaussian noise vector with zero mean and covariance matrix $\sigma^2\mathbf{I}_M$. A MIMO detector refers to an algorithm that is used to find an estimate $\hat{\mathbf{s}}$ of the transmitted symbol vector $\mathbf{s}$ when vector $\mathbf{x} = \mathbf{H}\mathbf{s} + \mathbf{n}$, as in (1), is received. In practice, this estimate is a set of LLR values to be fed to an outer decoder.

Maximum likelihood (ML) estimator is optimal in the sense of minimizing the error probability [6]. The ML solution can be computed using the equation [6, 37]

$$\hat{\mathbf{s}}_{\mathrm{ML}} = \arg\min_{\mathbf{s}\in\mathcal{C}} \|\mathbf{x} - \mathbf{H}\mathbf{s}\|^2, \tag{2}$$

where $\|\cdot\|$ denotes the Frobenius norm of a vector and $\mathcal{C}$ is the set of complex constellation points. For a system with N transmit antennas and constellation size $|\mathcal{C}|$, a total of $|\mathcal{C}|^N$ vector norms have to be calculated. As the number of transmit antennas or the constellation size increases, the computational complexity of a brute force maximum likelihood solution becomes quickly impractical [37].

Sphere detectors make it possible to find the ML or near-ML solution with reduced computational complexity. The basic idea is presented in Figure 1 using a demonstrative single-input single-output system with 64-QAM. The original constellation points are shown on the left as white circles, and the transmitted symbol is presented as a black circle. The channel skews the constellation lattice and noise is added to the received symbol (grey circle on the right) which now lies somewhere between the constellation points. Instead of a straightforward approach of computing the Euclidean distances between all possible symbols and the received symbol, the search is restricted inside a circle, and Euclidean distances are calculated only to those symbols that are inside the circle. Depending on the radius of the circle, the used constellation and the number of antennas, the approach can result in significant savings in computational complexity.

A list sphere detector (LSD) [17] is a sphere detector variant which, instead of giving just one most likely symbol vector, outputs a list of the most likely symbol vectors and their Euclidean distances. This modification makes the sphere detector suitable for soft-decision decoding, as shown in [17].

The K-best LSD [27], implemented in this paper, is a so called breadth-first algorithm which means that it processes the symbol levels one at a time. The idea is that at each level K best partial symbols with the smallest partial Euclidean distances (PEDs) to the received symbol are chosen to be

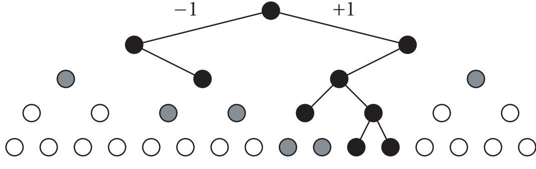
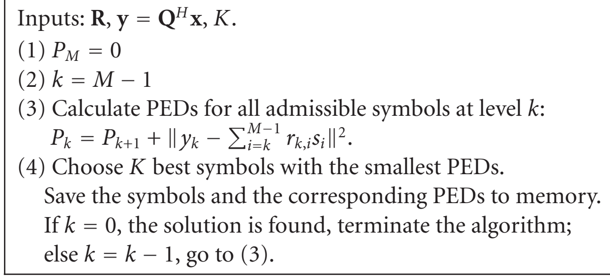


FIGURE 2: Tree presentations of the K-best LSD algorithm.



Inputs: $\mathbf{R}$, $\mathbf{y} = \mathbf{Q}^H\mathbf{x}$, K.
(1) $P_M = 0$
(2) $k = M - 1$
(3) Calculate PEDs for all admissible symbols at level k:
$P_k = P_{k+1} + \|y_k - \sum_{i=k}^{M-1} r_{k,i}s_i\|^2$.
(4) Choose K best symbols with the smallest PEDs.
Save the symbols and the corresponding PEDs to memory.
If $k = 0$, the solution is found, terminate the algorithm;
else $k = k - 1$, go to (3).

FIGURE 3: K-best LSD algorithm.

continued with. A graphical presentation of the K-best LSD algorithm for a 4×4 system using binary phase-shift keying (BPSK) modulation is shown in Figure 2. In BPSK, every transmitted symbol has only two possible values, -1 and $+1$. As this example is a real-valued 4×4 system, there are four levels in the tree. The highest circle in the tree is called the root node and it does not refer to any specific symbol. At the next layer there are two nodes (marked with circles) which represent the first symbol of the symbol vector $\mathbf{s}$ having the possible values of -1 and $+1$. When proceeding to the next levels, we can again choose between -1 and $+1$ until the bottom level is reached. As this is a BPSK example, the number of nodes at each level doubles every time we proceed to a lower level. The nodes at the lowest level are called leaf nodes and they correspond to all the 16 possible values of the symbol vector $\mathbf{s}$ from $[-1\ \ -1\ \ -1\ \ -1]^T$ to $[+1\ \ +1\ \ +1\ \ +1]^T$.

In Figure 2, $K = 2$ best nodes at each level are shown as black circles. The nodes whose PEDs have been calculated but that have been pruned because they did not succeed to be among the two smallest distances are shown as grey circles. Those nodes that have not been processed in any way are shown as white circles.

Mathematical foundations of sphere detection are presented in the following. The search can be limited inside a sphere with radius d using the sphere constraint:

$$d^2 \geq \|\mathbf{x} - \mathbf{H}\mathbf{s}\|^2. \tag{3}$$

The channel matrix $\mathbf{H}$ can be broken into two parts by using the QR decomposition. If the number of receive antennas equals the number of transmit antennas ($M = N$), the transformation can be presented simply as $\mathbf{H} = \mathbf{Q}\mathbf{R}$, where $\mathbf{R}$ is an $M \times M$ upper triangular matrix and $\mathbf{Q}$ is an $M \times M$ orthogonal matrix. Performing the QR decomposition we obtain

$$d^2 \geq \|\mathbf{x} - \mathbf{Q}\mathbf{R}\mathbf{s}\|^2 \Longleftrightarrow d^2 \geq \left\|\mathbf{Q}^H\mathbf{x} - \mathbf{R}\mathbf{s}\right\|^2. \tag{4}$$

The modified constraint can be further simplified by denoting $\mathbf{y} = \mathbf{Q}^H\mathbf{x}$ to get

$$d^2 \geq \|\mathbf{y} - \mathbf{R}\mathbf{s}\|^2. \tag{5}$$

Because of the upper triangular property of $\mathbf{R}$, (5) can be presented as

$$d^2 \geq \sum_{i=0}^{M-1}\left(y_i - \sum_{j=i}^{M-1} r_{i,j}s_j\right)^2. \tag{6}$$

Now the symbol vector components can be considered separately. The K-best algorithm processes one vector component first, chooses K best partial symbols and stores them. Next, those K best partial symbols are expanded to the next symbol level, and again K best partial symbols are chosen to be continued with until the whole symbol vector has been processed.

In our TTA implementation, the sphere radius was set to infinity, $d = \infty$, which guarantees a constant number of visited nodes in all cases. The K-best algorithm used in the implementation, modified from [38], is presented in Figure 3.

It is simple to decompose the complex-valued system model (1) into a real counterpart [23, 26, 27, 39]. This approach has some benefits, for example simple implementation of the Schnorr-Euchner enumeration (SEE), but it doubles the depth of the search tree which can be infavourable from the implementation point of view. In our work, we operate directly on the complex constellation points.

3. RELATED WORK

This section reviews some earlier work on sphere detector implementations including one field-programmable gate array (FPGA), one very long instruction word (VLIW) and several very-large-scale integration (VLSI) implementations.

The review is not limited to breadth-first or soft-output architectures only. The solutions that have been found for some specific sphere detector variant may be widely applicable in other architectures as well. All sphere detectors include PED calculation, and also some kind of a sorting algorithm is applied in many variants.

3.1. Early K-best VLSI architecture

The VLSI (ASIC) design in [27] can be considered as the starting point for sphere detector implementations. The design is based on using the real-valued decomposition of the system model. The architecture consists of tightly pipelined processing elements and is scalable to different numbers of antennas. The decoding throughput was estimated for a 4×4 system with 16-QAM and a list size of $K = 10$. The decoding order of symbols was assumed to be calculated before the detector for improved bit-error rate. The performance degradation with $K = 10$ was announced to be less than 0.5 dB at 20 dB SNR compared to the ML solution. The PED calculation is highly optimized, and the utilization of the functional units inside each processing element is said to be close to 100%. The whole K-best architecture consists

of approximately 52 000 gates, excluding the memory area of about 8600 bits. With 4×4 system and 16-QAM, a decoding throughput of 10 Mbps should be reached. The detector supports hard outputs.

The decoding throughput seems fairly good and a reasonable amount of gates is needed. However, the register-based bubble sort method needs $2 \times K - 1$ registers for every symbol stage where sorting is used. With long list lengths the amount of registers and their energy consumption would become impractical.

3.2. Two VLSI architectures for K-best Schnorr-Euchner enumeration (KSE)

In [26], two VLSI architectures for K-best Schnorr-Euchner enumeration are proposed. Both implementations decompose the 4×4 16-QAM system into real values and assume very efficient preprocessing before the decoding. The preprocessing takes the channel noise into account and orders the symbols for improved performance. In this way, the list size can be reduced down to 5 without suffering from too significant performance degradation. Bubble sort is used for choosing the best K symbols. The first version supports only hard outputs and is capable of a decoding throughput of 53.3 Mbps with approximately 91 000 gates. The second implementation supports soft outputs and uses a so called modified K-best Schnorr-Euchner enumeration (MKSE). MKSE tries to use the information contained in the discarded paths that can be virtually augmented to full length based on the assumptions about the remaining undetected symbols. One of the simplest ways to implement this is to use the ZF estimate. In this way, also the discarded paths can contribute to the soft-value generation. The soft-output MKSE achieves 106.6 Mbps decoding throughput with approximately 97 000 gates.

3.3. Two high-throughput complex-valued depth-first VLSI architectures

Two VLSI architectures with very high decoding throughputs are presented in [23]. Both implementations are based on processing the tree depth-first instead of the breadth-first approach used in the K-best algorithm. Both implementations, ASIC-I and ASIC-II, operate directly on the complex-valued constellation points which, according to the authors, leads to a more reasonable implementation. Both systems are designed for 4×4 antenna scheme and 16-QAM. The main differences between the two implementations are in their preprocessing strategies and in the realization of the Schnorr-Euchner enumeration. ASIC-II also uses a simplified L_∞ norm instead of the more common L_2 norm and thus cannot be considered an ML estimator any more. ASIC-I achieves a decoding throughput of 73 Mbps with approximately 117 000 gates. ASIC-II yields over doubled throughput of 169 Mbps with only less than half (50 000) gates compared to ASIC-I. Both throughputs are at 20 dB SNR. The performance degradation between ASIC-I and ASIC-II is told to be about 1.4 dB. Both implementations support hard outputs only.

3.4. Parallelized depth-first architecture

In [24], the hardware complexity is first investigated for exhaustive search ML estimation with different constellations and numbers of antennas. It is shown that up to 4×4 antennas with QPSK modulation, the exhaustive ML estimation is feasible. Beyond that, the complexity and power consumption increase dramatically. For example, full ML-APP (a posteriori probability) estimation for 4×4 16-QAM would yield almost 270 mm^2 area with 32.7 W power consumption which are obviously impractical values.

A depth-first list sphere detector is proposed as a more reasonable approach and an architecture is described for both the precomputation unit and the sphere detector. The precomputation unit computes the upper triangular decomposition of the channel matrix and the unconstrained ML estimate for the search center. For a 4×4 system and 16-QAM, a decoding throughput of 38.4 Mbps can be achieved with one precomputation unit and five parallel search engines and APP cost function units. The total area of the whole implementation is roughly estimated to be around 10 mm^2. The five parallel search engines take approximately 4.85 mm$^2 \times 1.3 \approx 6.3$ mm^2 of this total area (30% implementation overhead is assumed to account for items such as additional memories, clock trees). Gate counts are not presented, but assuming a gate density of 80 kgates/mm^2 that can be achieved with modern 0.18 μm technologies, the number of gates for five parallel search engines can be calculated as 6.3 mm$^2 \times 80$ kgates/mm$^2 \approx 500$ kgates.

3.5. K-best VLSI architectures achieving up to 424 Mbps

Two very-high-throughput VLSI architectures are presented in [39]. Both architectures output hard decisions and operate in a parallel and pipelined fashion. The second variation uses the simplified L_1 norm instead of the L_2 norm which is shown to lead to a significant reduction in circuit complexity but causing only a small bit-error rate (BER) performance loss. Both architectures use the real-valued decomposition.

The detectors are pipelined so that one layer of the tree is always processed in one pipeline stage. The architecture consists of metric computation units (MCUs) for PED calculation and K-best units (KBUs) that determine the K smallest PEDs and the corresponding symbol vectors. Register banks are used to store the K best symbols from the previous layers of the tree. The overall architecture consists of $2 \times N$ almost identical copies of pipeline stages, including the register bank, MCU and KBU, where N represents the number of transmit antennas.

After the real-valued decomposition is applied to a regular Y-QAM constellation, the $\sqrt{Y}$ new constellation points lie on the real axis. The MCU is used to compute PEDs for all possible $\sqrt{Y}$ children for some parent symbol. With very simple logic, it is possible for the MCU to output these values so that they are sorted. This feature is further exploited in the actual sorting unit, the KBU, where a simpler design can be used because of the presorted inputs.

The two proposed architectures are evaluated with two K values, 5 and 10, leading to four combinations. Using the simplified L_1 norm with $K = 5$, the highest published throughput to our knowledge, 424 Mbps, can be achieved. The core area of this architecture is estimated as 93 000 gates. If the channel can be assumed to remain the same for two subsequent received vectors, requiring the storage of one channel matrix only, the area can be reduced down to 68 000 gates.

3.6. FPGA implementation

FPGA implementation issues are considered in [40, 41]. Architectures were designed for a 2×2 system with QPSK and 16-QAM. The architectures are built of successive distance calculation and sorting blocks, and the sorting is handled with register-based sorting units. The 2×2 system running in QPSK mode and implemented on FPGA could reach a decoding throughput of 4.6 Mbps. If the same detector could be implemented as an ASIC, the throughput was estimated to rise to around 10 Mbps.

The detector complexity was estimated also for a 4×4 system and 64-QAM. However, throughput and gate count estimates are not available.

3.7. VLIW implementation

The same algorithm that was used in the TTA LSD implementation in this paper was programmed in C language and compiled for Texas Instruments C6711 digital signal processor in [42, 43]. The algorithm uses the complex-valued system model with 4×4 16-QAM, and heap structure is used to sort the symbol vectors with a list size of 63 items. The preprocessing part ($\mathbf{y} = \mathbf{Q}^H\mathbf{x}$) is not included in the TI implementation. Even though the processor is designed specifically for signal processing purposes, it is very understandable that it cannot achieve a reasonable decoding throughput especially as the processor does not have direct support for complex arithmetic. The processing time of one symbol vector was 293 000 clock cycles which is obviously too long for real-time applications as the maximum clock frequency of the processor is 150 MHz. A rough estimate is that this latency could be reduced with around 20% if the assembly code was optimized by writing it manually instead of using an optimizing compiler.

4. LSD IMPLEMENTATION ON ASIP

An ASIP was designed for K-best list sphere detector algorithm using the TTA computation paradigm, and the algorithm was implemented in TTA assembly.

In conventional architectures, data transports are consequences of operations whereas in TTA [31–33], the situation is reversed and operations are consequences of data transports. A TTA processor is programmed simply by defining the sources and destinations of these transports. For example, addition can be performed by moving the addends to the input ports of an addition unit and, on one of the following clock cycles, reading the result from the output port.

In this section, an overview of the processor implementation is given. The overall structure and operation, memory usage, and the PED unit of the processor are described in detail and several variations of the implementation are suggested.

In the following, the terms symbol vector and partial Euclidean distance (PED) may refer to either complete symbol vectors with four elements or partial symbol vectors with one to three elements, depending on the context. The symbol levels are referred to as levels 3, 2, 1, and 0, corresponding to the elements of the symbol vector, $[s_0 \quad s_1 \quad s_2 \quad s_3]^T$.

4.1. Implementation overview

A TTA processor for running the K-best LSD algorithm was designed for a 4×4 MIMO system. A complex-valued LSD variant that uses fixed-point arithmetic was used. A relatively long list size of 63 was chosen, and the storing and sorting of the symbol vectors was based on memory rather than registers. The detector was designed for 16-QAM. The K-best algorithm was implemented in TTA assembly so that it could be run on the designed processor.

A word length of 32 bits (16 bits for the real and 16 bits for the imaginary part) is used in computations including the elements of $\mathbf{Q}$, $\mathbf{x}$, $\mathbf{R}$, and $\mathbf{y}$. 16 bits were allocated for the PEDs.

The processor includes two load-store units (LSUs), two addition and subtraction units (ADDSUBs), one comparison unit (CMP) and one global control unit (GCU). Special function units (SFUs) were used for computing Euclidean distances and maintaining a list of the best candidate symbol vectors. In addition to the aforementioned building blocks, there is one general purpose register file that includes three 32-bit registers. The different parts of the processor are connected with ten buses, allowing a highly parallel operation. The processor architecture is presented in Figure 5.

4.2. Memory usage

The LSD algorithm processes combinations of symbol vectors and corresponding PEDs. As the implementation uses 16-QAM, every transmitted symbol, s_i, where $i = \{0, 1, 2, 3\}$, can be chosen from 16 different constellation points.

The symbols can be represented with binary numbers, 0000 corresponding to symbol $-3 - 3j$, 1111 corresponding to symbol $3 + 3j$, and so forth. Four bits are needed for each component of the symbol vector which adds up to 16 bits altogether as the symbol vector consists of four symbols as the complex-valued 4×4 system model was used. 16 bits were left for the Euclidean distance, resulting in 32 bits (one word) for the combination of the symbol vector and the corresponding Euclidean distance. This is illustrated in Figure 4.

The memory of the processor consists of 128 addresses that can all contain one memory word, thus leading to a total memory size of 4096 bits. Storing a list of 63 symbol candidates would need only 63 addresses, but the LSD algorithm needs to load previous level symbol vectors from the memory and use another memory area for sorting and storing the

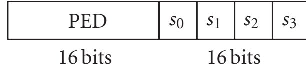


Figure 4: Storage format of PED and the corresponding symbol vector.

symbol vectors at the current level at the same time, so $2 \times n$ memory addresses are needed, where n equals the list size.

The memory can be thought to be divided in two equally sized areas, A and B. The LSD algorithm starts with computing PEDs for all of the possible 16 symbols at the third symbol level. Those symbol vectors and their PEDs are stored in the beginning of memory area B. As there are only 16 symbols, the symbols do not have to be sorted as $16 < 63 = K$. When the algorithm proceeds to the next level, there are $16 \times 16 = 256$ different symbol combinations, so sorting is needed. Now the symbols are read from memory area B and area A, after being reset, is used for sorting. Before proceeding to the next level, area B is reset. After that the previous level symbols are read from area A and B is used for sorting. Before proceeding to the final level to process the last component of the symbol vector, area A is reset, previous level symbols are read from B and A is used for sorting.

4.3. Sorting of symbol vectors

The K-best LSD algorithm maintains a list of K best symbol vectors that have the smallest Euclidean distances so far. If a large list size is preferred, the sorting and storing of symbol vectors quickly becomes the bottleneck of the algorithm. The list maintenance could be made really fast by using registers for sorting the list, but the register-based approaches tend to have too high energy consumption when a large list size is used even if the hardware is highly optimized.

Using memory instead of registers will provide a slower but possibly more energy-efficient solution to the problem. As the latency of inserting a new symbol to the list is very crucial for the overall performance of the LSD algorithm, an efficient data structure is needed for sorting and storing of symbols.

Heap data structure has been suggested for LSD algorithm already before [44, 45], but, according to our knowledge, implementations with detailed explanation of the heap utilization have not been published so far. Heap is an efficient choice for long lists as the complexity of insertion is only of order $O(\log_2 n)$ for binary tree-shaped heaps. Because of this low-order insertion complexity, the heap data structure was chosen for the implementation.

The heap is used with a custom-designed special function unit (SFU) that is used for address calculation and value determination. The SFU itself is used with a software algorithm so the list updating can be considered as an algorithm implemented in software but accelerated by an SFU, the list unit (LU).

The list unit for heap-based sorting is based on the unit described in [46], where also the heap data structure is presented in detail. The unit takes five inputs: the address of the current parent node, the data this address contains, the data that the child nodes of the parent node contain and the symbol level that is being processed. The unit decides whether the nodes should be swapped and outputs data that should be written to the current parent node and the child node that the parent node was possibly swapped with. In addition, it also gives the addresses of the new parent node and the new child nodes. The last input, `level`, is used for defining which memory area is used as a heap. The latency of the unit is one clock cycle.

The list insertion routine used in the implementation is able to insert a new symbol to the list in

$$C_{\text{insertion}} = 2\lceil \log_2(n+1) \rceil - 1 \tag{7}$$

clock cycles, where n equals the list size and $\lceil \cdot \rceil$ denotes rounding towards infinity (ceiling operation). With a list size of 63, $\lceil 2\log_2(63+1) \rceil - 1 = 11$ clock cycles are needed for each symbol insertion.

4.4. Pipelining of PED calculation and heap sorting

As presented above, inserting a new symbol in the heap takes 11 clock cycles. During this time, the PED of the next symbol can be computed.

At those symbol levels (2, 1, 0) where the heap is used for sorting the symbol vectors, one PED is calculated first. Then, at the same time when this PED is being inserted into the heap, the PED calculation starts for the next symbol in parallel with the insertion routine. This goes on until PEDs have been computed for all of the symbols on that level. After finishing the last PED computation, the last symbol is inserted in the heap.

4.5. The PED unit

An SFU was designed for the PED calculation also. The PEDs are calculated completely by this SFU and assembly routine is needed only for feeding the input values to the unit and reading the output value (PED) from it.

As the heap insertion has a constant duration of 11 clock cycles, the PED unit latency was constrained to be less than or equal to that. A more powerful PED unit for faster computation would not have given any benefit, so a low-complexity hardware unit with only one multiplication and one addition/subtraction unit could be designed.

The PED unit is capable of performing five different operations: *mmul*, *ped3*, *ped2*, *ped1*, and *ped0*.

The operation *mmul* is used for computing

$$\mathbf{y} = \mathbf{Q}^H \mathbf{x}. \tag{8}$$

Vector $\mathbf{y}$ is computed one element at a time, so matrix $\mathbf{Q}$ can be fed to the PED unit row by row instead of inputting the whole matrix (16 elements) at the same time. In this way, the number of input ports can be reduced. The values of $\mathbf{y}$ and $\mathbf{Q}$ are fed to the PED unit with 32 bit accuracy, using 16 bits for the real part and 16 bits for the imaginary part of each vector and matrix element.

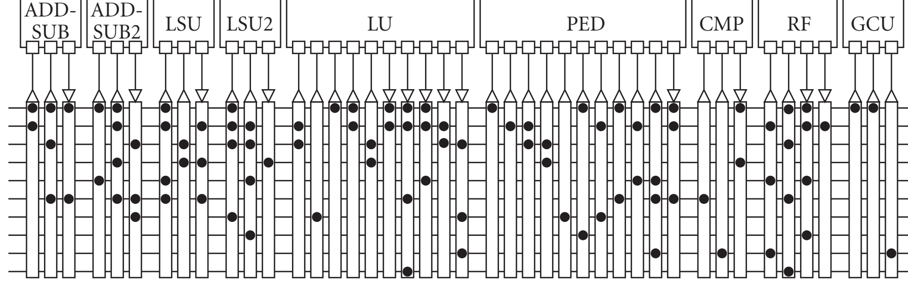


Figure 5: Processor architecture.

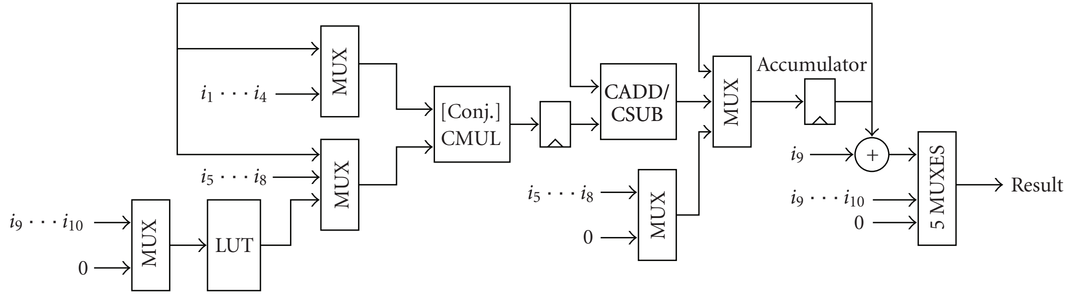


Figure 6: Block diagram of the PED unit.

The operation *ped3* is used for PED calculation at the third level. Breaking the summation presented in Figure 3 into its components gives for the third level

$$P_3 = 0 + \left|\left|y_3 - \sum_{i=3}^{3} r_{k,i}s_i\right|\right|^2 = ||y_3 - r_{3,3}s_3||^2. \tag{9}$$

The operation *ped2* is used for PED calculation at the second level. Similarly to *ped3*, the summation now gives

$$\begin{aligned} P_2 &= P_3 + \left|\left|y_2 - \sum_{i=2}^{3} r_{k,i}s_i\right|\right|^2 \\ &= P_3 + ||y_2 - r_{2,2}s_2 - r_{2,3}s_3||^2. \end{aligned} \tag{10}$$

The operation *ped1* is used for calculating

$$\begin{aligned} P_1 &= P_2 + \left|\left|y_1 - \sum_{i=1}^{3} r_{k,i}s_i\right|\right|^2 \\ &= P_2 + ||y_1 - r_{1,1}s_1 - r_{1,2}s_2 - r_{1,3}s_3||^2. \end{aligned} \tag{11}$$

The operation *ped0* is used for the last PED calculation:

$$\begin{aligned} P_0 &= P_1 + \left|\left|y_0 - \sum_{i=0}^{3} r_{k,i}s_i\right|\right|^2 \\ &= P_1 + ||y_0 - r_{0,0}s_0 - r_{0,1}s_1 - r_{0,2}s_2 - r_{0,3}s_3||^2, \end{aligned} \tag{12}$$

The squared magnitude of a complex number can be computed using multiplication as

$$\|w\|^2 = w \cdot w^*, \tag{13}$$

where w^* denotes the complex conjugate of $w \in \mathbb{C}$. The PED unit is designed so that the internal complex multiplication unit can perform both normal multiplications and multiplications where one of the multiplicands is conjugated.

The unit has ten input ports whose purposes depend on the operation that is executed. For *mmul* operation, the first eight inputs are used for inputting the values of matrix $\mathbf{Q}$ and vector $\mathbf{x}$. For the PED operations, the first four inputs are used for feeding the elements of the $\mathbf{R}$ matrix, and the next four inputs are used for inputting the vector $\mathbf{y}$. In addition, the symbol vector from the previous level with its corresponding PED and the current level symbol are input to the ninth and tenth input ports, respectively.

The internal functionality of the PED unit is described with a simplified block diagram in Figure 6. The values of the input registers of the PED SFU are denoted by i_x, where $x = \{1, 2, 3, \ldots, 10\}$ and multiple input ports of multiplexers with $i_a \cdots i_b$, where $a, b = \{1, 2, 3, \ldots, 10\}$. The first multiplexer before the look-up table (LUT) selects alternative bit slices of the inputs i_9 and i_{10}, extracting the symbols and the corresponding previous level PED from the inputs. The look-up table is used for transforming the symbols from the four-bit format to a format that is suitable for complex-valued multiplications. Between the complex multiplier and the complex addition/subtraction unit there is one register stage. The PED unit contains an internal accumulator whose initial value can be set according to the current operating mode. The last adder before the last multiplexers is a real-valued adder that is used for adding the contribution of the current symbol to the previous level PED. The last five multiplexers compose the final result by combining the intermediate values bitwisely.

In principle, the PED SFU multiplexes the same computing resources to compute the desired results sequentially. Such an approach requires accurate control of the computing resources and intermediate results. Multicycle operations are controlled with the aid of an internal counter which keeps track of the operation steps. According to the operation code and the value of the counter, a control word that controls all the multiplexers and arithmetic operations is formed. The generation of the control word is not shown in Figure 6.

4.6. Variations of the implemented version

To explore the possibilities for performance enhancements, three different variations are proposed. Their effects to latency and hardware complexity are estimated in Section 5.

4.6.1. Software-pipelined heap insertion

Another heap utilization strategy that reduces the clock cycles to $\log_2(n+1)+1$ per insertion was presented in [46]. The insertion latency approaches the theoretical limit of heap insertion complexity ($O(\log_2(n))$) when $n\to\infty$. With a list size of 63, the insertion latency can be dropped down from 11 to $\log_2(63+1)+1 = 7$ clock cycles.

4.6.2. Conditional jump out of the insertion routine

Version A

As explained in Section 4.3, the insertion routine of the implemented version always lasts for 11 cycles which allows a low-complexity PED unit. However, the routine could be modified for higher throughput by enabling a conditional jump out of the insertion routine. By adding a simple comparator to the processor, the insertion routine could detect on the first clock cycle of insertion whether the new candidate fits in the heap or not. If the candidate is larger than the heap maximum, a jump instruction could be executed on the first clock cycle already. Because of jump latency of four clock cycles, there would still be four clock cycles executed in the routine even if the candidate did not fit in the heap.

Now the PED would have to be computed in three clock cycles for it to be ready before the possible jump. In the implemented version, the insertion latency as well as the latency of the whole LSD algorithm is constant, whereas enabling the conditional jump would make the insertion and LSD latencies variable.

Version B

Using conditional jump out of the insertion routine could be implemented in another way also. An additional output port could be included in the list unit, see [46]. If the new symbol does not fit in the heap or the nodes are not swapped at some point during the insertion routine, the unit could detect this and generate an output value, `continue`. The conditional jump could be made by using guarded execution, and the jump could be executed on the second clock cycle of the insertion routine. Also in this version the PED computation would have to be faster than in the implemented version. However, the latency demands are not as strict as for Version A, and a PED latency of five clock cycles could be accepted.

4.6.3. Parallel processing of five symbol vectors

As can be seen from summations (9)–(12), the complexity of PED calculation varies from level to level. Using a conditional jump out of the insertion routine asks for faster PED computation as the computation has to be timed so that it is ready even if the insertion routine is interrupted. This means parallel multiplications and subtractions inside the PED unit for all symbol levels except for the first one and, of course, the need for parallel arithmetic operations requires more robust hardware. However, a hardware unit that is able to perform four multiplications and subtractions simultaneously has purposeless resources when considering the PED calculation at easier levels. Also, using a highly parallel PED unit for computing $\mathbf{Q}^H\mathbf{x}$ is not efficient. This inefficiency that originates from using the same unit for operations that require different amounts of hardware resources could be avoided by implementing five different computation units: one for computing $\mathbf{Q}^H\mathbf{x}$ and four units for PED calculation on different symbol levels. These units could be used to process five symbol vectors parallelly, leading to higher throughput and more efficient use of resources.

4.7. More efficient PED calculation

The partial Euclidean distance calculation that is needed in the implemented sphere detector is a demanding procedure that includes complex multiplications, subtractions, and squaring operations.

In the following, three simple modifications are presented that could be used to achieve more efficient PED calculation at the expense of increased design complexity.

Breaking the PED unit into smaller parts

The possibility to map the PED calculation functionality to one unit greatly simplifies the algorithm at assembly-level. The whole computation with possibly several multiplications and subtractions, bitwise operations and squared magnitude calculations can be executed with one simple instruction, which allows relatively straightforward assembly-level implementation. However, to utilize the hardware resources even more efficiently than the implemented version does, the PED unit could be broken into smaller parts that could be used in a more pipelined way. However, the design complexity would increase significantly what comes to assembly-level programming, and some kind of custom-made function units would still be necessary to accelerate the PED calculation.

Precomputing the PED partially for one common parent symbol

The efficiency of the PED computation could be improved in another way also. A closer look at the PED calculation (see (9)–(12)) reveals that many of the multiplications could be done at once for one parent symbol [39]. Precomputing a

Table 1: Processor building blocks and their estimated areas at 100 MHz clock frequency using 0.13 μm technology.

Unit	Operation(s)	Area/Gates
RF	Register load, store	1600
ADDSUB	Addition, subtraction	1100
CMP	Equal, signed/unsigned greater	1100
LSU	Memory load, store	600
PED	PED computation	8900
LU	List unit	2300
SWLU	Software-pipelined list unit	2800

part of the PED in advance for one parent symbol would leave simplified computation to be done for the children symbols, leading to simpler hardware.

Simpler multiplications with constellation points

The fact that many of the multiplications in the LSD algorithm have a constellation point as one multiplicand could be used to reduce the hardware complexity. Full complex-valued multiplications with two variable operands and squaring operations have high circuit complexity while multiplications with constellation points have negligible circuit complexity that is comparable to adders [23].

5. LATENCY AND HARDWARE COMPLEXITY ESTIMATION

In this section, the latencies and data path complexities of different possible designs are estimated and compared to each other. Also the effects of reduced list size and parallelization are investigated as possibilities for achieving higher decoding throughput.

The area estimates consider the data path complexity first. The additional area requirements that come from, for example, the control logic and interconnection network, are first neglected but their effect is discussed later. Exact latency is provided from simulation results for the implemented version. The other variations are characterized by their total heap insertion latencies which give fairly good estimates of the overall latencies.

5.1. *The implemented version*

The latency of the implemented version is constant as the insertion routine always takes exactly 11 clock cycles and also the number of heap insertions is constant. The heap insertion is used $16 \times 16 + 16 \times 63 + 16 \times 63 = 2272$ times. The total insertion latency (clock cycles) can be calculated as

$$C_{\text{impl}} = 2272 \times 11 = 24992. \tag{14}$$

However, some additional clock cycles will come from, for example, controlling the program flow, performing the matrix multiplication $\mathbf{Q}^H\mathbf{x}$ in the beginning of the algorithm, computing PEDs for the third level symbols and resetting the heap with maximum values during program execution. The simulation results for the implementation show that the complete execution of the algorithm takes 26400 clock cycles. As $(26400/24992 - 1) \times 100\% \approx 5.6\%$, the overhead that comes from other operations than running the insertion routine can be considered relatively small. This justifies using the total insertion latencies of different variations as a good starting point for comparing them with each other.

Different building blocks of the processor were modeled in very-high-speed integrated circuit hardware description language (VHDL) and synthesized with Synopsys Design Compiler for area estimates. Table 1 shows the estimated areas (gate counts) of different basic building blocks and the SFUs that were used in the implementation, synthesized with 0.13 μm technology at 100 MHz clock frequency. Also the list unit for software-pipelined execution is included in the table, and the operations that the different units support are presented for clearness. The register file is assumed to include three 32-bit registers with two input and two output ports.

Considering that the implementation consists of two ADDSUB units, two LSUs, a CMP unit, an RF, a PED unit, and an LU, the area (number of gates) can be estimated as

$$\begin{aligned} G_{\text{impl}} &\approx 2 \times 1100 + 2 \times 600 + 1100 \\ &\quad + 1600 + 8900 + 2300 = 17300. \end{aligned} \tag{15}$$

5.2. *Software-pipelined heap insertion*

The amount of heap insertions remains the same (2272 insertions) if software-pipelined heap utilization is used. However, the time per insertion drops down from 11 clock cycles to seven and the basic software-pipelined version would have a constant insertion latency (clock cycles) of

$$C_{\text{SW}} = 2272 \times 7 = 15904. \tag{16}$$

An area estimate can be calculated like for the implemented version, taking into account that six LSUs are needed instead of just two. The list unit for software-pipelined execution (SWLU) is also slightly more complex than in the implemented version, see Table 1. In addition, more performance is required from the PED unit also as the computation has to be finished a little earlier. The capability for simultaneous subtractions is needed inside the PED unit which is taken into account by adding the term 200 that approximates this complexity increase. The gate count can be estimated as

$$\begin{aligned} G_{\text{SW}} &\approx 2 \times 1100 + 6 \times 600 + 1100 + 1600 \\ &\quad + (8900 + 200) + 2800 = 20400. \end{aligned} \tag{17}$$

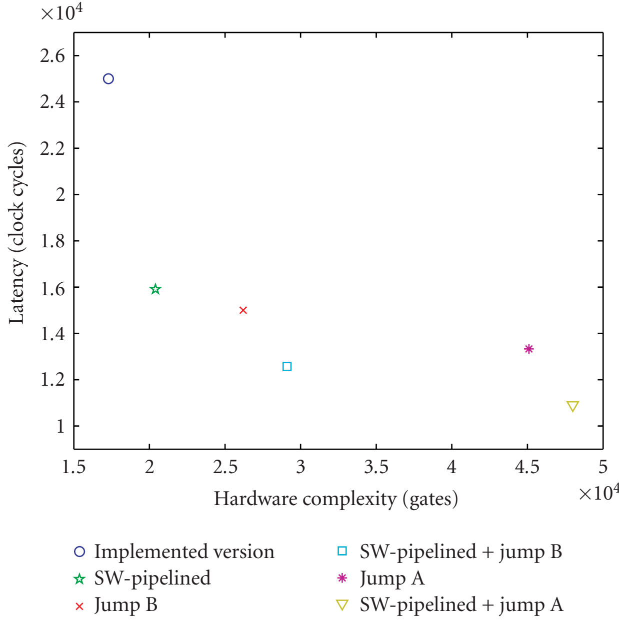


Figure 7: Latency and hardware complexity estimates of the implementation and the proposed variations.

5.3. Conditional jump out of the insertion routine

Version A

With the first version (Version A) of the conditional jump out of the insertion routine, proposed in Section 4.6.2, the insertion routine latency would be either four or 11 clock cycles. If all of the PEDs of inserted symbols are assumed to have equal probability distributions, simple simulations can be made to estimate how many inserted symbols will fit in the heap (leading to 11-cycle insertion) and how many will not (leading to four-cycle insertion). On level 2 (256 insertions, heap size 63 items), about 40% of the symbol candidates will fit in the heap after it has been initially filled. At levels 0 and 1 (1008 insertions, heap size 63 items), only about 18% of the symbols will fit in the heap after initial filling. Using these assumptions, the average latency can be estimated as

$$\begin{aligned} C_{\text{jump_A}} = 11 \times (&63 + 63 + 63 + 0.4 \times (256 - 63) \\ &+ 0.18 \times 2 \times (1008 - 63)) \\ + 4 \times (&0.6 \times (256 - 63) + 2 \times 0.82 \\ &\times (1008 - 63)) = 13332.8. \end{aligned} \quad (18)$$

The increase in hardware complexity is significant compared to the implemented version. One comparator unit has to be added, but that is not the main reason for higher complexity. The fact that the PED calculation has to be done in three clock cycles requires a highly parallel PED unit. The unit has to be able to perform four complex multiplications during one clock cycle. Also four subtractions have to be computed simultaneously. Assuming that the size of the parallel PED unit is quadrupled from the basic PED unit used in the implementation, the gate count of the architecture is

$$\begin{aligned} G_{\text{jump_A}} \approx\ & 2 \times 1100 + 2 \times 600 + 2 \times 1100 + 1600 \\ & + 4 \times 8900 + 2300 = 45100. \end{aligned} \quad (19)$$

Version B

Using the second jump strategy would lead to a slightly longer latency than the first approach. However, the hardware requirements are a lot more relaxed. For simplicity, we assume that the insertion routine would last for either five or 11 clock cycles, depending on the situation as for version A. Now the overall insertion latency can be estimated using the same assumptions as for Version A as

$$\begin{aligned} C_{\text{jump_B}} = 11 \times (&63 + 63 + 63 + 0.4 \times (256 - 63) \\ &+ 0.18 \times 2 \times (1008 - 63)) \\ + 5 \times (&0.6 \times (256 - 63) + 2 \times 0.82 \\ &\times (1008 - 83)) = 14998.4. \end{aligned} \quad (20)$$

The reason for less strict hardware requirements is the fact that additional CMP unit is not needed and that the PED unit latency can be as large as five clock cycles now, leading to a smaller PED unit. The gate count of the PED unit is approximated to double from the basic PED unit as two parallel multiplications and subtractions are needed. The hardware complexity of the list unit is assumed to be equal to that of the list unit used in the implemented version. The complexity increase from adding one new output port (`continue`) is negligible as existing control signals can be used for determining the output value. The gate count can be estimated as

$$\begin{aligned} G_{\text{jump_B}} \approx\ & 2 \times 1100 + 2 \times 600 + 1 \times 1100 + 1600 \\ & + 2 \times 8900 + 2300 = 26200. \end{aligned} \quad (21)$$

5.4. Comparison of alternative TTA processors

The conditional jump out of the insertion routine can naturally be applied to both the software-pipelined and nonpipelined design. Combining different strategies, six different schemes can be considered:

(i) implemented version,
(ii) jump A,
(iii) jump B,
(iv) software-pipelining,
(v) software-pipelining + jump A,
(vi) software-pipelining + jump B.

The latency and area estimates for the first four designs were presented above. It is easy to estimate the last two latencies using the same principles as before.

Figure 7 shows a graphical presentation of the different alternatives. The data path hardware complexities (gate counts) and total insertion latencies of different approaches are compared. It can be seen immediately that utilizing the first jump strategy (Version A) without software-pipelined heap insertion is not a reasonable option in any case as

smaller latency can be achieved with simpler hardware with software-pipelined insertion and jump version B. Also the high latency of the implemented version is quite obvious, and significant improvements can be achieved by utilizing the proposed alternatives without too noticeable increases in hardware complexity.

Figure 7 alone is not enough to put the proposed variations in order in terms of efficiency. As everything else except the FUs and the register file is neglected in the area estimates, a constant term has to be added to them. The efficiency order of different designs depends on the area of the excluded hardware including the GCU, interconnection network, control logic, memories, and so forth. The excluded area can be thought of as an unavoidable cost that has to be added to build a functional processor. Separating the data path complexity from the rest of the hardware has some benefits. The approach allows clear comparisons between different processor alternatives as the additional costs are only weakly dependent on the data path complexity of the design. In addition, comparison to pure hardware solutions is straightforward. Also, if the designed functionality was to be added to an existing processor, the data path complexity would be the most interesting part.

The whole processor, including the datapath, control logic and interconnection network was synthesized with $0.13\,\mu$m technology at 100 MHz and it required approximately 26 600 gates, excluding the memory. The proportional part of the datapath compared with the overall processor core area can be calculated approximately as $(17300/26600) \times 100\% \approx 63\%$.

5.5. *Reducing the list size for higher throughput*

The architecture was designed to enable long lists without utilizing an impractical amount of registers that have high power consumption. The basic design principle in this work was to process and sort the symbol vectors sequentially. Even with a highly optimized software-pipelined heap insertion routine, the total latency of the algorithm will remain too high to achieve a practical decoding throughput with reasonable clock frequencies and processor areas if a list size as large as 63 is used.

Assuming efficient preprocessing (e.g., optimal ordering of the processed symbols) and high SNR, the list size could be reduced. If a list size of seven items was used, the latency of each insertion would be only $\log_2(n+1)+1 = \log_2(7+1)+1 =$ 4 clock cycles. In addition to this, the amount of insertions would be reduced to $16+7\times16+7\times16+7\times16 = 352$. This would lead to an insertion latency of $4 \times 352 = 1408$ clock cycles. Compared with the overwhelming 15904 clock cycles that is to be faced when software-pipelined insertion is used with 63 items, the speedup is significant as the processing time can be reduced with $(1-(1408/15904))\times100\% \approx 91\%$. And still one has to notice that the 1408 clock cycles already include the insertion of 16 symbol vectors at the third symbol level which is excluded from the 15904 clock cycles as with $K \geq 16$, sorting is not needed at the third symbol level.

However, some overhead has to be added to the number of pure insertion cycles, and the latency of the whole algorithm could be roughly approximated as 1500 clock cycles, see Section 5.1 for overhead estimation. If the processing of five symbol vectors was parallelized, the average time for processing one symbol vector could be reduced down to about 1500/5 = 300 clock cycles. In a 4×4 system with 16-QAM, 16 coded bits are transmitted in every symbol vector as one 16-QAM symbol carries four bits and there are four transmit antennas. At 100 MHz clock frequency, a throughput of approximately $16/(300/(100 \times 10^6))$ Mbps $\approx$ 5.3 Mbps could be achieved.

Rough estimates can be made about the hardware complexity of the proposed parallel architecture. We assume that one symbol vector can be processed with the hardware for software-pipelined heap utilization, see Section 5.2, but now including a PED unit whose gate count is doubled from the PED unit used in the implemented version so that the PEDs can be computed in four clock cycles. Multiplying the required hardware by five, we may approximate the datapath complexity of the parallelized architecture as around 145500 gates.

Additional gates would be needed for additional hardware resources, including the control logic and interconnection network. In the implemented version, this area was estimated as about 9300 gates (see Section 5.4). Assuming that this additional area would remain the same as for the implemented version and adding 10% implementation overhead, a rough estimate for the total gate count can be made as $(145\,500 + 9300)$ gates $\times\, 1.1 \approx 170$ kgates.

5.6. *Discussion*

Precise comparisons between different sphere detector architectures is practically impossible as different designs may perform differently in different channels conditions, with different antenna spacing, at different SNR, and so forth. Also the design complexity and the flexibility of the design always affects the usefulness of some specific idea. However, the basic facts about the reviewed 4×4 16-QAM designs are summarized in Table 2. The reference is given along with K (if used), the system model (real-valued or complex-valued), decoding throughput (T), and gate equivalent (GE) number estimates.

Software-pipelined heap insertion and conditional jump out of the insertion routine were shown to offer higher decoding throughput without increasing the hardware complexity too significantly. A list size of 63 seems to be impractical, and a reduced list size is proposed to enable real-life implementation.

It is obvious that even the parallelized TTA processor proposal with reduced list size is not able to rival the fast register-based ASIC implementations in terms of throughput if the sequential processing strategy is used. However, the ASIP design that was presented in this paper has several advantages over the fixed ASIC implementations. The detector could operate with a smaller number of antennas just by modifying the program that is executed. Also the list size could be reduced programmably to speed up and simplify the processing in high SNR where it is possible to maintain a reasonable BER level with a shorter list. The possibility to adapt the list size would allow adjusting the amount

Table 2: Comparison of different sphere detector architectures with 4×4 system and 16-QAM.

Architecture	K	System	T/Mbps	kGE
Parallel TTA processor	7	C	5.3	170
Early VLSI [27]	10	R	10	52
KSE, [26]	5	R	53.3	91
MKSE [26]	5	R	106.6	97
ASIC-I [23]	—	R	73	117
ASIC-II [23]	—	R	169	50
Parallel depth-first [24]	—	C	38.4	500
424 Mbps [39]	5	R	424	93/68

of computation under different circumstances so that the detection could be performed with a minimal energy consumption.

If the values of the constellation points can be assumed to remain the same when changing to a lower-order QAM constellation, the detector would be applicable also to QPSK and BPSK without any modifications. (By constant values we mean that when changing from 16-QAM to QPSK or BPSK, the constellation points that the detector operates on are changed from $\{-3-3j,\ldots,3+3j\}$ to $\{-1-j,\ldots,1+j\}$ or $\{-1,1\}$, resp.) This kind of flexibility is something that, to the best of our knowledge, has not been presented in any of the ASIC publications.

The proposed design contains several individual, standard units (two addition-subtraction units, a comparator, general-purpose registers, load-store units) that could be used also for other applications than just list sphere detection. Also the sorting functionality could be suitable for other applications where sorting is needed. The PED unit for distance calculation could be applied at least in some other sphere detector variants. Naturally, the processor does not share the general-purpose nature of a conventional DSP, but compared to ASICs it would still allow lots of possibilities just by modifying the executed program.

With very short lists, it is likely that a simple, purely hardware-based sorting method can provide the best energy-efficiency. However, the basic idea of our approach was to design a processor that would be scalable to be used with long or very long lists as well. In our design, the insertion latency grows logarithmically with the list size, and the required hardware remains completely unchanged if the increased, but still modest, memory size requirements are neglected. In register-based ASICs, increasing the list size increases the required hardware complexity, and with very long lists, the solutions would become extremely impractical in terms of hardware complexity and energy consumption. This is probably why the ASIC implementations do not seem to address solutions that would enable even intermediate list sizes (the largest list size among the ASICs included in our review was $K = 10$).

6. CONCLUSIONS

This paper began by giving an overview of MIMO detection algorithms with most weight on K-best sphere detection. The earlier work on sphere detector implementations was presented, and a programmable ASIP design for K-best LSD was presented and described in detail. The design space was explored by presenting and evaluating several modifications that could be used for improving the decoding throughput.

To the best of our knowledge, the presented K-best implementation is the first published ASIP design for sphere detection. In addition, the memory-based heap sort method used in the implementation opens completely new perspectives for low-power sphere detector design.

Future research topics could include a K-best LSD implementation with software-pipelined heap insertion routine and even more fine-tuned PED calculation, where different symbol levels could be considered in parallel with specially designed PED units. Also the possibility of reaching a higher decoding throughput with register-based sorting methods should be considered. Future work should concentrate also on investigating efficient methods for enabling small list sizes. With the current technology, reaching real-time performance with a list size of about 64 seems impractical with any implementation technique. The next implementation should be combined with performance simulations including iterative channel coding, realistic channel models and precise word length studies.

ACKNOWLEDGMENT

This work has been supported by the Finnish Funding Agency for Technology and Innovation, Nokia Siemens Networks, Elektrobit, and Texas Instruments.

REFERENCES

[1] I. E. Telatar, "Capacity of multi-antenna gaussian channels," Internal Technical Memorandum, pp. 1–28, Bell Laboratories, Suffolk, UK, 1995.

[2] E. Telatar, "Capacity of multi-antenna gaussian channels," *European Transactions Telecommunication*, vol. 10, pp. 585–595, 1999.

[3] G. J. Foschini and M. J. Gans, "On limits of wireless communications in a fading environment when using multiple antennas," *Wireless Personal Communications*, vol. 6, no. 3, pp. 311–335, 1998.

[4] P. W. Wolniansky, G. J. Foschini, G. D. Golden, and R. A. Valenzuela, "V-BLAST: an architecture for realizing very high

data rates over the rich-scattering wireless channel," in *Proceedings of the International Symposium on Signals, Systems and Electronics, (ISSSE '98)*, pp. 295–300, Pisa, Italy, 1998.

[5] D. Gesbert, M. Shafi, D Shiu, P. J. Smith, and A. Naguib, "From theory to practice: an overview of MIMO space-time coded wireless systems," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 3, pp. 281–302, 2003.

[6] A. J. Paulraj, D. A. Gore, R. U. Nabar, and H. Bölcskei, "An overview of MIMO communications—a key to gigabit wireless," *Proceedings of the IEEE*, vol. 92, no. 2, pp. 198–217, 2004.

[7] H. Boelcskei, D. Gesbert, C. B. Papadias, and A. J. van der Veen, *Space-Time Wireless Systems: From Array Processing to MIMO Communications*, Cambridge University Press, Cambridge, UK, 2006.

[8] M. Myllylä, J. M. Hintikka, J. Cavallaro, M. Juntti, M. Limingoja, and A. Byman, "Complexity analysis of MMSE detector architectures for MIMO OFDM systems," in *Conference Record—Asilomar Conference on Signals, Systems and Computers*, vol. 2005, pp. 75–81, Pacific Grove, Calif, USA, 2005.

[9] H. Artés, D. Seethaler, and F. Hlawatsch, "Efficient detection algorithms for MIMO channels: a geometrical approach to approximate ML detection," *IEEE Transactions on Signal Processing*, vol. 51, no. 11, pp. 2808–2820, 2003.

[10] G. D. Golden, C. J. Foschini, R. A. Valenzuela, and P. W. Wolniansky, "Detection algorithm and initial laboratory results using V-BLAST space-time communication architecture," *Electronics Letters*, vol. 35, no. 1, pp. 14–16, 1999.

[11] U. Fincke and M. Pohst, "Improved methods for calculating vectors of short length in a lattice, including a complexity analysis," *Mathematics of Computation*, vol. 44, pp. 463–471, 1985.

[12] O. Damen, A. Chkeif, and J. C. Belfiore, "Lattice code decoder for space-time codes," *IEEE Communications Letters*, vol. 4, no. 5, pp. 161–163, 2000.

[13] H. Yao and G. W. Wornell, "Lattice-reduction-aided detectors for MIMO communication systems," in *Proceedings of the IEEE Global Telecommunications Conference*, vol. 1, pp. 424–428, Taipei, Taiwan, 2002.

[14] D. Wübben, R. Böhnke, V. Kühn, and K. Kammeyer, "Near-maximum-likelihood detection of MIMO systems using MMSE-based lattice-reduction," in *Proceedings of the IEEE International Conference on Communications*, vol. 2, pp. 798–802, Paris, France, 2004.

[15] M. O. Damen, H. El Gamal, and G. Caire, "On maximum-likelihood detection and the search for the closest lattice point," *IEEE Transactions on Information Theory*, vol. 49, no. 10, pp. 2389–2402, 2003.

[16] P. Silvola, K. Hooli, and M. Juntti, "Sub-optimal soft-output MAP detector with lattice reduction," *IEEE Signal Processing Letter*, vol. 13, pp. 321–324, 2006.

[17] B. M. Hochwald and S. Ten Brink, "Achieving near-capacity on a multiple-antenna channel," *IEEE Transactions on Communications*, vol. 51, no. 3, pp. 389–399, 2003.

[18] Y. L. C. de Jong and T. J. Willink, "Iterative tree search detection for MIMO wireless systems," *IEEE Transactions on Communications*, vol. 53, no. 6, pp. 930–935, 2005.

[19] J. W. Kang and K. B. Lee, "Simplified ML detection scheme for MIMO systems," in *Proceedings of the IEEE Vehicular Technology Conference*, vol. 2, pp. 824–827, Milan, Italy, 2004.

[20] B. Hassibi and H. Vikalo, "On the sphere-decoding algorithm I. expected complexity," *IEEE Transactions Signal Processing*, vol. 53, pp. 2806–2818, 2005.

[21] H. Vikalo and B. Hassibi, "On the sphere-decoding algorithm II. generalizations, second-order statistics, and applications to communications," *IEEE Transactions Signal Processing*, vol. 53, pp. 2819–2834, 2005.

[22] D. C. Garrett, L. M. Davis, and G. K. Woodward, "19.2 Mbit/s 4×4 BLAST/MIMO detector with soft ML outputs," *Electronics Letters*, vol. 39, no. 2, pp. 233–235, 2003.

[23] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bölcskei, "VLSI Implementation of MIMO detection using the sphere decoding algorithm," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 7, pp. 1566–1576, 2005.

[24] D. Garrett, L. Davis, S. Ten Brink, B. Hochwald, and G. Knagge, "Silicon complexity for maximum likelihood MIMO detection using spherical decoding," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 9, pp. 1544–1552, 2004.

[25] D. Garrett, G. K. Woodward, L. Davis, and C. Nicol, "A 28.8 Mbit/s 4×4 MIMO 3G CDMA receiver for frequency selective channels," *IEEE Journal of Solid-State Circuits*, vol. 40, pp. 320–3302, 2005.

[26] Z. Guo and P. Nilsson, "Algorithm and implementation of the K-best sphere decoding for MIMO detection," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 3, pp. 491–503, 2006.

[27] K. Wong, C. Tsui, R. K. Cheng, and W. Mow, "A VLSI architecture of a K-best lattice decoding algorithm for MIMO channels," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 3, pp. 273–276, Scottsdale, Ariz, USA, 2002.

[28] C. Schlegel and L. Prez, *Trellis and Turbo Coding*, Wiley IEEE Press, Piscataway, NJ, USA, 2004.

[29] W. G. Jeon, K. H. Chang, and Y. S. Cho, "Instrumentable tree encoding of information sources," *IEEE Transactions on Information Theory*, vol. 17, no. 1, pp. 118–119, 1971.

[30] J. Anderson and S. Mohan, "Source and channel coding: an algorithmic approach," *IEEE Transactions on Communications*, vol. 32, pp. 169–176, 1984.

[31] H. Corporaal, *Microprocessor Architectures: From VLIW to TTA*, John Wiley & Sons, New York, NY, USA, 1998.

[32] H. Corporaal, "Design of transport triggered architectures," in *Proceedings of the 4th Great Lakes Symposium on (VLSI '94)*, pp. 130–135, Notre Dame, Ind, USA, 1994.

[33] H. Corporaal, "A different approach to high performance computing," in *Proceedings of the 4th International Conference on High Performance Computing*, pp. 22–27, Bangalore, India, 1997.

[34] J. Antikainen, P. Salmela, O. Silvén, M. Juntti, J. Takala, and M. Myllylä, "Transport triggered architecture implementation of list sphere detector," in *Proceedings of the Finnish Signal Processing Symposium*, Oulu, Finland, August 2007.

[35] J. Antikainen, P. Salmela, O. Silvén, M. Juntti, J. Takala, and M. Myllylä, "Application-specific instruction set processor implementation of list sphere detector," in *Proceedings of the 39th Annual Asilomar Conference on Signals, Systems Composition*, Pacific Grove, Calif, USA, 2007.

[36] 3rd Generation Partnership Project, "Group radio access network requirements for evolved UTRA (E-UTRA) and evolved UTRAN (E-UTRAN)," Technical Specification TR 25.913 version 7.3.0 (release 7), 3rd Generation Partnership Project, Valbonne, France, 2006.

[37] T. Fujita, T. Onizawa, W. Jiang, D. Uchida, T. Sugiyama, and A. Ohta, "A new signal detection scheme combining ZF and

K-best algorithms for OFDM/SDM," in *Proceedings of the IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, (PIMRC '04)*, vol. 4, pp. 2387–2391, 2004.

[38] M. Myllylä, P. Silvola, M. Juntti, and J. R. Cavallaro, "Comparison of two novel list sphere detector algorithms for MIMOOFDM systems," in *Proceedings of the IEEE International Symposium Personal, Indoor, Mobile Radio Communications*, pp. 12–16, Helsinki, Finland, September 2006.

[39] M. Wenk, M. Zellweger, A. Burg, N. Felber, and W. Fichtner, "*K*-best MIMO detection VLSI architectures achieving up to 424 Mbps," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1151–1154, Kos, Greece, 2006.

[40] J. Kerttula, "Implementation of a *K*-best based multiple antenna detector," M.S. thesis, Department of Electrical and Information Engineering, University of Oulu, Oulu, Finland, 2007.

[41] J. Kerttula, M. Myllylä, and M. Juntti, "Implementation of a *K*-best based MIMO-OFDM detector algorithm," in *Proceedings of the European Signal Proccessing Conference*, Poznán, Poland, 2007.

[42] J. Janhunen, "Signal processor implementation of list sphere detection," M.S. thesis, Department of Electrical and Information Engineering, University of Oulu, Oulu, Finland, 2007.

[43] J. Janhunen, O. Silvèn, M. Myllylä, and M. Juntti, "A DSP implementation of a *K*-best list sphere detector algorithm," in *Proceedings of the Finnish Signal Processing Symposium*, p. 6, Oulu, Finland, 2007.

[44] A. Wiesel, X. Mestre, A. Pags, and J. R. Fonollosa, "Efficient implementation of sphere demodulation," in *Proceedings of the IEEE Workshop on Signal Processing Advances in Wireless Communications*, pp. 36–40, Rome, Italy, June 2003.

[45] B. Widdup, G. Woodward, and G. Knagge, "A highlyparallel VLSI architecture for a list sphere detector," in *Proceedings of the IEEE International Conference on Communications*, pp. 2720–2725, Paris, France, June 2004.

[46] P. Salmela, J. Antikainen, O. Silvén, and J. Takala, "Memory-based list updating for list sphere decoders," in *Proceedings of the IEEE Workshop on Signal Processing Systems (SiPS '07)*, pp. 633–638, Shanghai, China, 2007.