Improving Packet Processing Performance in the ATLAS FELIX Project

Analysis and Optimization of a Memory-Bounded Algorithm

Jörn Schumacher[*]
CERN, Geneva, Switzerland
and
University of Paderborn,
Germany
joern.schumacher@cern.ch

ABSTRACT

Experiments in high-energy physics (HEP) and related fields often impose constraints and challenges on data acquisition systems. As a result, these systems are implemented as unique mixtures of custom and commercial-off-the-shelf electronics (COTS), involving and connecting radiation-hard devices, large high-performance networks, and computing farms. FELIX, the Frontend Link Exchange, is a new PC-based general purpose data routing device for the data-acquisition system of the ATLAS experiment at CERN. Performance is a very crucial point for devices like FELIX, which have to be capable of processing tens of gigabyte of data per second. Thus it is important to understand the performance limitations for typical workloads on modern hardware. In this paper the analysis of FELIX packet processing algorithm is presented. The role played by the PC system's memory architecture in the overall data throughput is discussed and motivated, both by measurements and theoretical means. Finally, optimizations increasing the processing throughput by a factor larger than 10x are analyzed.

1. INTRODUCTION

The ATLAS experiment [1] is one of the four experiments at the LHC, the Large Hadron Collider, at CERN, Geneva, Switzerland. During operation, ATLAS generates data at O(100 Tb/s), mostly from well known physics phenomena. Hence only a small fraction is stored for offline analysis. Of the 40 MHz collision event rate that the LHC generates, only about 1 kHz is preserved. It is the task of the trigger and data acquisition system to process and filter events in quasi real-time, before they are written to permanent storage.

The trigger and data acquisition system is organized in layers. The L1 trigger system reduces the 40 MHz collision rate to roughly 100 kHz. As the L1 trigger needs to accept or reject events within 2.5 microseconds, it is entirely implemented using custom electronic components. Accepted event fragments are forwarded from the on-detector electronics to the back-end electronics (ReadOut Drivers, RODs), located in a separate service cavern, over custom point-to-point detector-specific links. The RODs perform data manipulation tasks like aggregation or compression before pushing the data to the circa 100 ReadOut System (ROS) PCs over 1800 point-to-point optical links (S-Link.) ROS PCs buffer event fragments and forward them upon request to the High-Level Trigger (HLT) computer farm, consisting of 1500 servers. Here, the event rate is further reduced to the target event rate of about 1 kHz.

The FELIX project is a novel approach in interfacing the various ATLAS detectors to the data-acquisition system. Beginning in the next long shutdown of the LHC in 2018, various detector links will be gradually replaced by links based on the Versatile Link [2] and GBT [3] projects of CERN. The idea of FELIX is to provide a device with data routing capabilities interfacing the new detector high-throughput data-links directly to a switched COTS network. This means that the current layer of point-to-point connections is replaced with a switched network. FELIX allows the use of dynamic routing rules that enable load balancing, failure tolerance and easier data-flow management. FELIX systems are intended to be partially integrated in ATLAS first in 2018; a full deployment is planned for 2025. An overview of the architectural changes of the ATLAS data-acquisition system with FELIX is given in Figure 1.

A typical FELIX connects to 24–48 optical detector links operating at a data rate of 3.2 Gbps each. On the network, side 40 Gbps Ethernet links can be used, but other high-performance network technologies are also considered.

1.1 FELIX Implementation

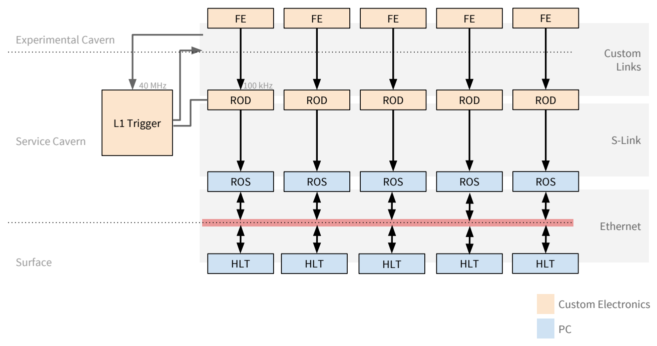
FELIX interfaces the detectors to the data-acquisition system, but also integrates monitoring, control, and calibration systems via the switched network. This is shown in Figure 2.

In the current development FELIX is based on COTS PC-hardware. The detector links connect to a PCIe Gen-3 FPGA-based card, while 40 Gbps Ethernet NICs provide the needed network connectivity (Figure 3).

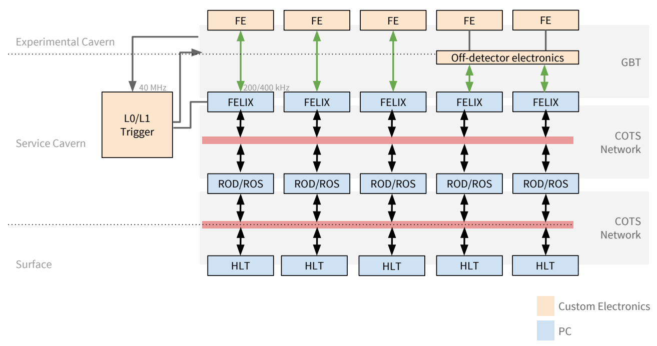
An application runs on the FELIX hosts to read and write data on the detector links. It manages network connections,

[*]Corresponding author, co-authors are listed in section 6.

DEBS '15, June 29 - July 3, 2015, OSLO, Norway.
ACM 978-1-4503-3286-6/15/06.
http://dx.doi.org/10.1145/2675743.2771824.



(a) Current Design



(b) Anticipated architecture for LHC Run-IV (2025)

Figure 1: ATLAS data-aquisition architecture with and without FELIX.

Throughput per Link	3.2 Gbps
Links per interface card	24
Links per system	24–48
Throughput per system	10–20 GB/s

Table 1: FELIX demonstrator requirements

routing data to one or more network locations based on extracted meta-information. The application uses a pipeline architecture with different processing steps, see Figure 4.

The large amount of data generated by the ATLAS experiment imposes certain requirements. Almost 10000 detector links will have to be connected to FELIX systems. To ensure a dense, cost- and space-efficient system, each FELIX will need to interface as many links as possible. The project demonstrator, which is estimated for Q1/2015, is designed to interface at least 24 detector links. If each of these links operates at a data rate of 3.2 Gb/s, 8B/10B encoded, the FELIX firmware and software will need to be able to process data at roughly 6.5 GB/s after 8B/10B decoding. Additionally, data packets cannot be delayed infinitely and should be forwarded within a short time frame. The requirements for a FELIX system are also listed in Table 1.

1.2 Contribution

The focus of this publication is the specific part of the FELIX software that handles the decoding of data packets transmitted over the PCIe bus by the link interface card. This piece of software plays a crucial role in the overall

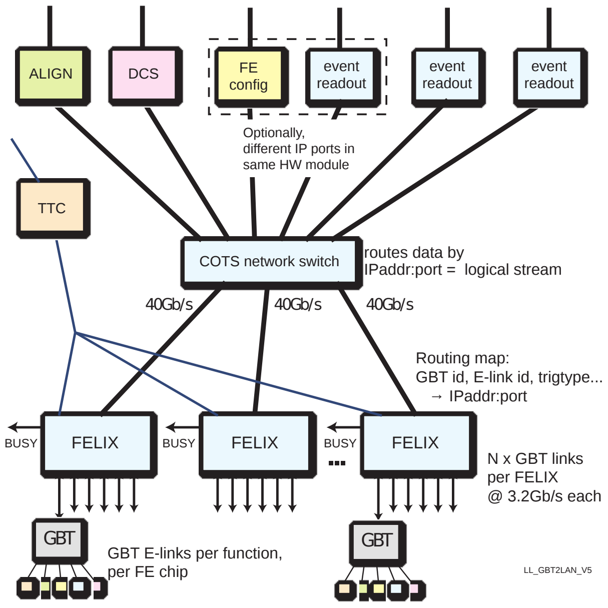

ALIGN
DCS
FE config
event readout
event readout
event readout
Optionally, different IP ports in same HW module
TTC
COTS network switch
routes data by IPaddr:port = logical stream
40Gb/s
40Gb/s
40Gb/s
Routing map: GBT id, E-link id, trigtype... → IPaddr:port
BUSY
FELIX
BUSY
FELIX
BUSY
FELIX
N x GBT links per FELIX @ 3.2Gb/s each
GBT
GBT E-links per function, per FE chip
GBT
LL_GBT2LAN_V5

Figure 2: FELIX integrates in the ATLAS data-acquisition system. On-detector electronics is connected via bi-directional Versatile links (labelled GBT). Readout, monitoring and detector control systems (DCS) are connected via commercial network technology. FELIX also manages LHC clock information (TTC: Timing, Trigger & Control).

performance of the FELIX application. The paper introduces first the data encoding for the PCIe transfers and the software decoding algorithm. The performance optimizations and results of such a packet processing algorithm are then discussed. Finally, further analysis of the software will demonstrate that memory bandwidth is the primary bottleneck limiting the algorithm speed.

2. ANALYSIS OF PACKET PROCESSING PERFORMANCE

In this section the evolution of the FELIX packet processing algorithm is present. Guided by profiling tools, a simple initial algorithm is optimized to be able to cope with the throughput requirements of the FELIX use case.

2.1 The Packet Processing Algorithm

In the normal use case, a packet-based transfer protocol is used on top of the physical detector links. Packets can have variable length; their content is not standardized and depends on the source. Usually packet boundaries are defined by an 8B/10B encoding or a similar technique. It is the task of the link interface card firmware to decode the packet stream and transmit packets, which are called *chunks* in FELIX terminology, over the PCIe bus into the host system's memory.

For technical reasons of the FPGA firmware, in order to move chunks from the PCIe link interface card, the chunks are packed into fixed-size *blocks*. Every block has a 4 byte header (see Table 2a), which encodes a 2 byte start-of-block word, a 5 bit sequence number and an 11 bit data stream

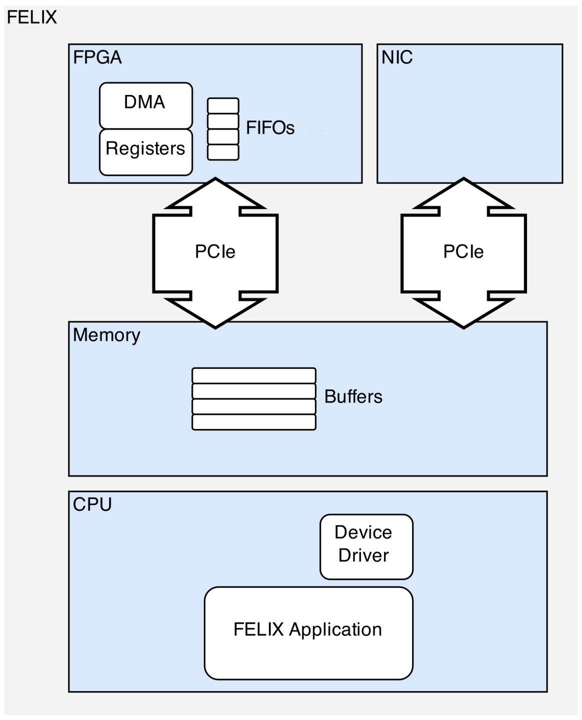


Figure 3: The architecture of a FELIX system. Multiple large buffers are used to enable multithreading.

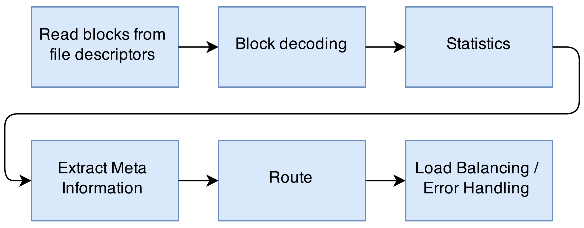

Read blocks from file descriptors
Block decoding
Statistics
Extract Meta Information
Route
Load Balancing / Error Handling

Figure 4: FELIX application pipeline. File descriptors expose the link interface card. Each file descriptor is associated to one large data buffer.

Bit range	Description
0-10	Stream ID
11-15	Sequence Number
16-31	Start-of-Block Symbol (0xABCD)

(a) Block Header (4 byte)

Bit range	Description
0-9	Length in Byte
10	Reserved for length field extension
11	Chunk error bit
12	Truncation bit
13-15	Type field

(b) Subchunk Trailer (2 byte)

Table 2: The meta-data in block headers and subchunk trailers included in the packets transmitted over PCIe.

identifier. The variable-length chunks are split into so-called *subchunks* to fit into the fixed-size blocks. Every subchunk has a 2 byte trailer including the length encoded as 10 bit integer, a truncation bit, an error bit, and a 3 bit field indicating the type of this subchunk (Table 2b). The subchunk type can either be *first*, *last*, or *middle*, indicating this subchunk starts a new chunk, ends a chunk or is in the middle of a chunk, *both*, indicating that this subchunk represents a full chunk that has not been split up, *null*, indicating that this subchunk does not carry data and is only used to fill up the block, or *out-of-band*, indicating that the rest of the trailer is to be interpreted as an out-of-band signal. The block format is illustrated in Figure 5.

The algorithm starts processing the subchunks at the end of a block. The subchunk trailer is read and a pointer to the data part of this subchunk is stored in a stack data structure. When a full chunk has been read, the pointers on the stack are read in reverse order and stored in a data structure. Note that only pointers to the actual data are stored. Using scattered read and write routines (readv, writev on POSIX) the data can be copied into a consecutive memory region or, for example, passed on to a network card, enabling a zero-copy application design.

2.2 Profiling

For the performance measurements, test data were generated with mixed-size chunks and processed with the packet processing algorithm isolated and in-memory.

The Intel VTune Performance Analyzer utility [4] was used to perform an initial profiling of an algorithm execution. Several issues were revealed by the profile and could be fixed. See the next section for details on optimizations.

As a next step, VTune was used to measure memory transactions while the benchmark was running. The results suggested a high CPI (clocks per instruction) of more than 2.5 in parts of the code as well as a large number of LLC (last-level cache) misses. The high number of LLC misses was expected since the benchmark was designed to read block data from main memory, as in a real-world scenario where data is copied to main memory via PCIe. The high CPI rate suggests that instructions are stalling and ILP (instruction-level parallelism) cannot be used effectively. This indicates the limited memory bandwidth of the test system to be the performance bottleneck.

2.3 Benchmark Results

Benchmarks were performed on two different test systems, see Table 3. System 1 is a single-socket system with 4 cores and a modern CPU. System 2 is a dual-socket system with more cores than System 1, but a slower memory. Results are presented in Figure 6.

2.4 Optimizations

The first optimizations were guided by results of the VTune profile. The usage of STL containers could be improved by avoiding unnecessary allocations, reserving memory upfront, constructing objects in-place (requires compiler support for C++11), and trying different backends for the stack data structure (see 2.1).

The runtime profile revealed that the usage of the stack data structure is relatively expensive. In some cases, like subchunks of type *both*, which represent a whole complete chunk, the stack can be avoided entirely. Changing the im-

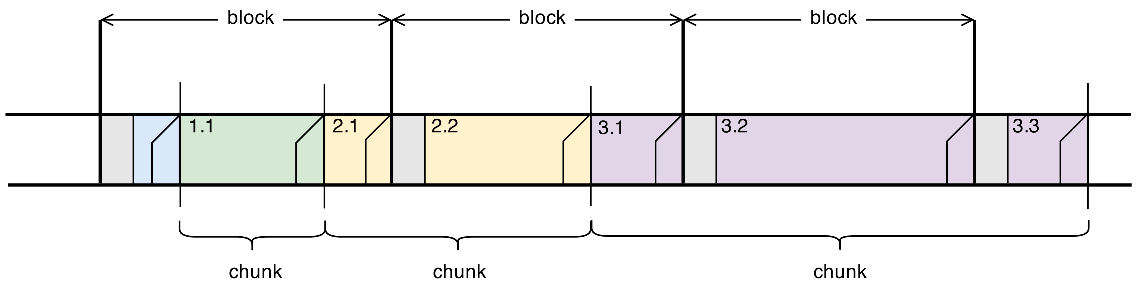


Figure 5: The block data format used to transmit data over the PCIe bus. The numbers indicate subchunks and their containing chunks, e.g., chunk 2 consists of two subchunks 2.1 and 2.2. Each block starts with a 4 byte header (left-most rectangle in each block), each subchunk ends with a 2 byte trailer (slanted shape at the end of each subchunk).

plementation to omit the stack in these cases could improve the runtime significantly.

Other optimizations involved compiler option tuning, the usage of NUMA-aware memory allocations and core-pinning to ensure that memory accesses are always local, and experiments with data prefetching using SSE intrinsics. These optimizations are represented by the "optimized" line in Figure 6.

The packet processing benchmark results show that it takes much longer to process a block containing many small chunks than a block containing few, but larger chunks (see Figure 6, "baseline"). This is expected since the amount of processing and data acesses increases when more trailers have to be parsed. On the other hand, the chance that a short chunk has to be split up in several subchunks is much smaller than for a large chunk. For example, 15 chunks with a size of 64 bytes fit into the 1020 bytes payload of a block without being split up into subchunks. As a result, only a single data pointer has to be stored. For small chunk sizes, this situation is common enough to have a dedicated specialized implementation. A new data type for short chunks was introduced which only contains a single data pointer. Construction of this object is significantly faster compared to a variable-length lists of pointers. There are two different data types now: (i) a complex data type consisting of a variable-length array of pointers to subchunk data and sizes, and (ii) a simple data type consisting of a single pointer and a length field. As a result, the processing speed for blocks with short chunks is reduced significantly (see Figure 6, "optimized, new data type").

The optimizations discussed in the previous sections reduce the processing time per block significantly. The speedup is larger for small chunk sizes, as some of the optimizations are specifically targeted for this scenario, but also for larger chunk sizes the processing time could be reduced. Speedups of above 10x are achieved on both test systems.

Figure 6 shows a plot of the average processing time per fixed-size block assuming that only chunks of equal size are stored in blocks. For each data point 100 MB of chunk data were generated and encoded in the fixed-size block encoding described earlier. Note that the average processing time for the System 1 is in all cases better than for the System 2. As will be discussed in the next section this can be attributed to the different memory speeds of the systems.

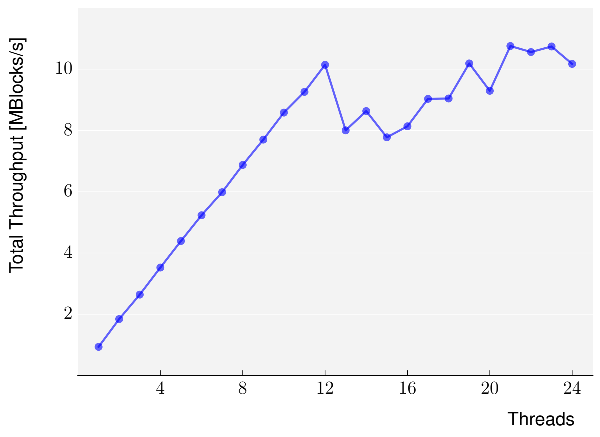
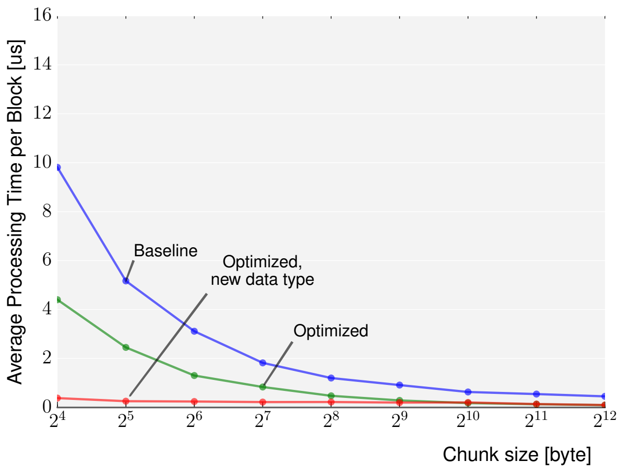


Figure 7: Overall block processing throughput for System 2 for different numbers of threads. The speedup is linear in the number of threads as long as HyperThreading is not used; the throughput saturates for more than 12 threads.

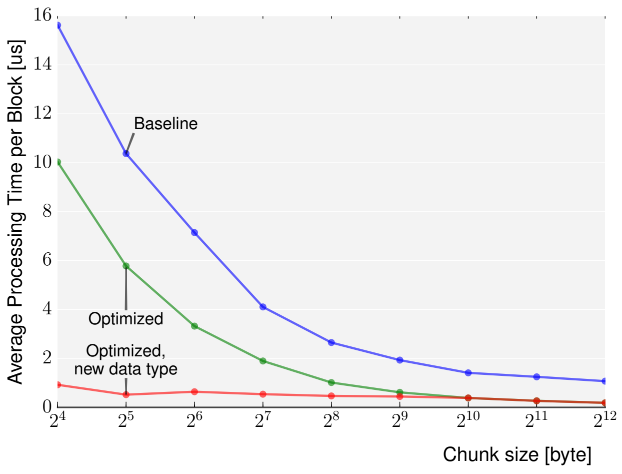
In a second experiment the influence of multithreading on the performance was analyzed. In the FELIX application, packet processing is embarrassingly parallel since the link interface card firmware is designed to support multiple buffers each thread can operate on a separate buffer. In the experiment this scenario is emulated by starting multiple threads, each working on an independent buffers of test data. Similarly to the previous experiment each thread is given 100 MB of block data to process. Results are shown in Figure 7. The speedup is almost linear in the number of threads used, with the exception of System 2, where the speedup is less than linear for more than 12 threads due to HyperThreading. With this system it is possible to process more than 10^6 blocks per second, which is more than the minimum throughput threshold for 24 Links at 3.2 Gb/s each (8B/10B encoded data), i.e. the amount of links foreseen to be connected per link interface card.

	System 1	System 2
CPU Type	Intel Core i7-3770	Intel Xeon E5645
CPU Clock Speed	3.40 GHz	2.40 GHz
Instruction Set Extensions	SSE4.1/4.2, AVX	SSE4.2
Nr of cores	4	12 (24 with Hyperthreading)
Nr of CPUs	1	2
Memory	8 GB DDR3 @ 1600 MHz	24 GB DDR3 @ 1333 MHz
Nr of Memory Modules	2	6 (3 per CPU)

Table 3: Specifications of the systems used for benchmarks.



(a) System 1



(b) System 2

Figure 6: Average runtime per processed block for two different test systems and for different stages of the optimization process. Note that System 1 has faster memory modules.

3. MEMORY BANDWIDTH ANALYSIS

In this section a more in-depth analysis of the memory-access aspects of the implementation is presented. First, the memory access pattern of the decoding algorithm is characterized and compared to a memory benchmark with a similar access pattern. Second, a Roofline model analysis is performed to determine the bottleneck of the algorithm by theoretical means.

3.1 Memory Access Throughput

The PMBW [5] benchmark collection was used to characterize the test systems for different memory access patterns and test scenarios. PMBW allocates buffers of different sizes and processes these buffers using different routines with different memory access patterns. The benchmarks include several sequential scanning and random access routines. The results for System 1 can be seen in Figure 8. System 2 behaves similar but is slightly slower. For single-threaded scans from main memory 5–10 GB/s were measured on System 2, compared to 10-20 GB/s for the same scenarios on System 1.

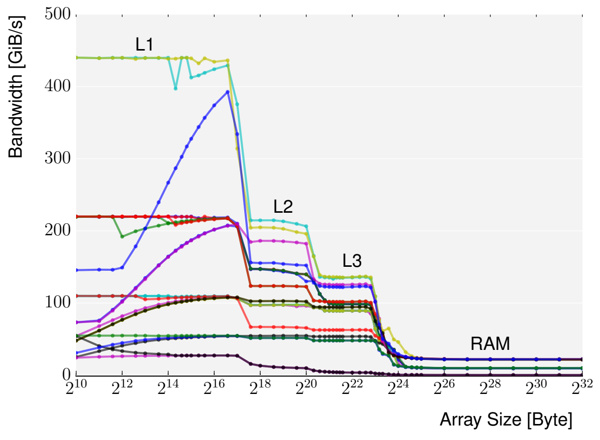
The effects of caching are visible in the PMBW benchmarks: test scenarios with small buffer sizes benefit from the differently sized CPU caches. On the other hand, caching can be ignored in FELIX, since data are copied via PCIe to main memory. Therefore scenarios with large buffers that are fully stored in main memory will be used for the following discussion.

The memory access pattern in the packet processing algorithm consists of many short reads of 2 bytes for the subchunk trailers and fewer reads of 4 bytes for the chunk headers. Chunks are read sequentially, but since only trailers and headers are processed, large parts of the data are skipped. This particular access pattern is similar to the Scan/Read scenarios with short data lengths in PMBW. These are depicted in Figure 8c.

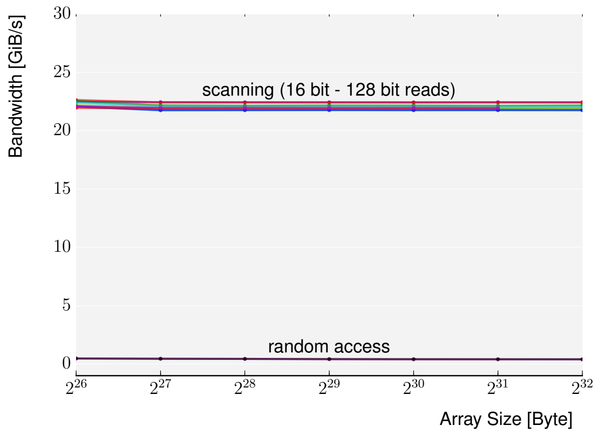
One can see that better memory performance would be possible with a different memory access pattern, for example with reads of more than 16 bit, implying changing the data format of the block encoding. On the other hand, the current algorithm is significantly faster than scenarios with a complete “random” memory access.

The measured read bandwidth during the packet processing benchmark on System 1 was between 8 and 9 GB/s in the optimized version. This is slightly less than the peak bandwidth of ca. 11GB/s obtained by PMBW, single-threaded Scan/Read/32Bit/SimpleLoop for this access pattern.

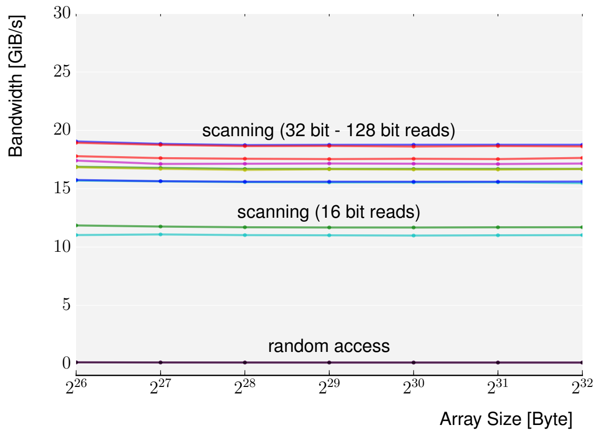
The memory access pattern results in a large number of cache misses. In a typical scenario with relatively short chunks the majority of reads will be 16 bit reads for subchunk trailers. In most modern x86-based CPU architectures memory is always read in 64 byte cachelines. As a result, to process one chunk, always a whole cacheline has to be read, even though only 2 bytes (the subchunk trailer) are used. The memory read efficiency is therefore only about 1/32.



(a) All memory layers, 4 threads



(b) Main memory transfers, 4 threads



(c) Main memory transfers, 1 thread

Figure 8: An evaluation of test System 1's memory performance using PMBW. In (a) the effects of the different cache levels are clearly visible. The packet processing algorithm accesses data from main memory, which is shown in (b) and (c). The benchmark uses different algorithms per read size to assess the memory performance. Memory speed does not multiply with the number of threads used.

3.2 Roofline Model Analysis

The Roofline model, as described in [6], is a modelling method used to describe the performance of an algorithm implementation in the context of limited memory bandwidth and computing speed. It is useful to identify bottlenecks and can give directions for optimization.

For the Roofline model, the performance P of an algorithm implementation is measured and related to its operational intensity I. The operational intensity is a property of the implementation and measures the average amount of instructions that are issued per byte read from memory. The measured performance P is then compared to two performance ceilings, the memory ceiling and the compute ceiling. Implementations with a low operational intensity are limited by the memory ceiling, whereas implementations with a high operational intensity are limited by the compute ceiling.

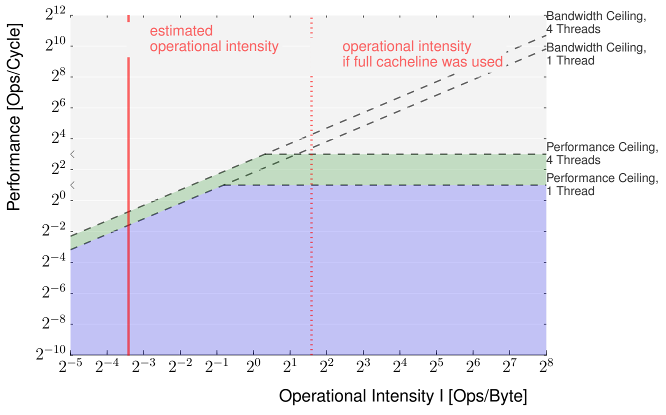
In the case of the packet processing algorithm the operational intensity was approximated by counting the number of operations that are needed to process one subchunk trailer, and dividing this number by the amount of memory that has to be read for the computation. The subchunk trailer is 2 byte long, but as indicated before a whole 64 byte cacheline must be read in order to process the 2 bytes. The operation count is estimated to be 6 operations per trailer, thus $I = 6/64 = 0.09375$ Ops/byte. The memory ceiling is measured by the PMBW benchmarks, the performance ceiling is estimated as 2 Ops/Cycle per thread. This assumes pipelined integer operations with a 2-fold instruction-level parallelism. The Roofline model analysis of the packet processing algorithm for chunks of 64 bytes is depicted in Figure 9.

The algorithm is clearly bounded by memory. This is expected since the algorithm is computationally not very demanding, but has many memory accesses and cannot benefit from caches, and thus has a low operational intensity. An increased operational intensity would therefore increase the measured performance. This could for example be achieved by an improved data layout. If the link interface card would store subchunk trailers not interleaved with data, but in a separate meta-data table, multiple subchunk trailers can be read at once when accessing a cacheline. This hypothetical scenario is indicated in Figure 9 by the dotted red line. According to the model, this optimization would shift the algorithm nearly into the compute-bound region. However, implementing this optimization would require support in the firmware of the link interface card. It is also not clear that the speedup would be as indicated by the model, since it would also require changes to the algorithm, and therefore to the number of operations needed.

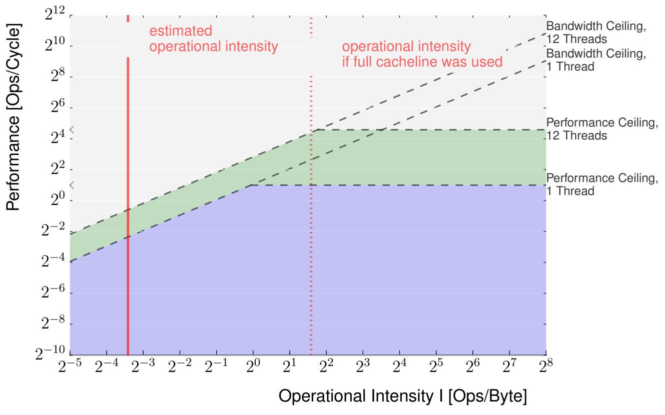
4. CONCLUSION

In this paper, FELIX, a new data distribution and routing device for the ATLAS experiment at CERN, was introduced. The experiment operation imposes challenging requirements on this system, especially in terms of throughput.

The implementation of packet processing algorithm compatible with the FELIX requirements required several levels of optimizations. Advanced profiling tools were fundamental in achieving the necessary throughput performance. Furthermore, it was demonstrated that the resulting algorithm is limited by the test system memory bandwidth.



(a) System 1



(b) System 2

Figure 9: A Roofline model analysis of the packet processing algorithm for 64 bytes long chunks. Due to the low read efficiency (only 2 of 64 bytes read are actually used for processing) the benchmark is limited by the memory speed.

In order to independently validate this result, a Roofline model analysis was performed. This confirmed that the FELIX packet processing algorithm is memory-bounded. This analysis also provided additional insights on the Roofline model. While it is certainly useful, the Roofline model can only be seen as a first-order approximation, especially effective in classifying an implementation as memory-bound or compute-bound. Quantities like the operational intensity are hard to obtain, by measurement or just plain code analysis. Moreover, as today's CPUs get more and more complex and include features like ILP, pipeline architectures or micro-ops, it is hard to give a good estimate of a CPU's peak performance.

The FELIX demonstrator will be based on an Intel Haswell architecture and DDR4 RAM technology with significantly faster data transfer rates than the benchmark systems discussed in this paper. Since the memory performance is a primary driving factor of the packet processing performance, newer hardware is expected to provide an increased throughput. Based on the results from this paper we estimate that on the new hardware the packet processing performance will continue growing linearly with the memory performance.

5. ACKNOWLEDGMENTS

Work supported by the Wolfgang-Gentner-Programme of the Bundesministerium für Bildung und Forschung (BMBF).

6. ADDITIONAL AUTHORS

J. T. Anderson (Argonne National Laboratory), A. Borga (Nikhef National Institute for Subatomic Physics and University of Amsterdam), H. Boterenbrood (Nikhef National Institute for Subatomic Physics and University of Amsterdam), H. Chen (Brookhaven National Laboratory), K. Chen (Brookhaven National Laboratory), G. Drake (Argonne National Laboratory), D. Francis (CERN), B. Gorini (CERN), F. Lanni (Brookhaven National Laboratory), G. Lehmann Miotto (CERN), L. Levinson (Weizmann Institute of Science), J. Narevicius (Weizmann Institute of Science), C. Plessl (University of Paderborn), A. Roich (Weizmann Institute of Science), S. Ryu (Argonne National Laboratory), F. P. Schreuder (Nikhef National Institute for Subatomic Physics and University of Amsterdam), W. Vandelli (CERN), J. Vermeulen (Nikhef National Institute for Subatomic Physics and University of Amsterdam), J. Zhang (Argonne National Laboratory).

7. REFERENCES

[1] The ATLAS Collaboration. The ATLAS Experiment at the CERN Large Hadron Collider. *Journal of Instrumentation*, 3(08):S08003, 2008.

[2] L Amaral, S Dris, A Gerardin, T Huffman, C Issever, A J Pacheco, M Jones, S Kwan, S-C Lee, Z Liang, et al. The versatile link, a common project for super-LHC. *Journal of Instrumentation*, 4(12):P12003, 2009.

[3] P Moreira, R Ballabriga, S Baron, S Bonacini, O Cobanoglu, F Faccio, T Fedorov, R Francisco, P Gui, P Hartin, K Kloukinas, X Llopart, A Marchioro, C Paillard, N Pinilla, K Wyllie, and B Yu. The GBT Project. *Topical Workshop on Electronics for Particle Physics*, pages 342–346, 2009.

[4] Intel Corporation. Intel VTune Amplifier 2015. https://software.intel.com/en-us/intel-vtune-amplifier-xe.

[5] T Bingmann. Parallel Memory Bandwidth Benchmark. http://panthema.net/2013/pmbw/.

[6] S Williams, A Waterman, and D Patterson. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, April 2009.