GIVING TEXT ANALYTICS A BOOST

Traditional server architectures cannot analyze big data efficiently. By using a streaming hardware accelerator implemented in reconfigurable logic, the authors can improve the throughput of SystemT’s information extraction queries by an order of magnitude. Such a system can be deployed by extending SystemT’s existing compilation flow and using a multithreaded communication interface that can utilize the accelerator’s bandwidth.

Information extraction is the task of extracting desired information from textual data and transforming it into a table. Several frameworks, such as the General Architecture for Text Engineering (GATE) and the Natural Language Toolkit (NLTK), address this task. IBM’s SystemT software couples a declarative rule language with a modular runtime based on relational algebra, augmented with special operators for information-extraction primitives such as regular expressions and gazetteers. This improves the rule language’s expressive power while also enabling cost-based rule optimization, improving extraction throughput.

SystemT software uses a document-per-thread execution model that enables each software thread to work on an independent document in parallel—similar to the approach taken by the GATE software through the GATECloud.net service, which enables deployment of an annotation pipeline on a compute cloud. Measurements have shown up to a tenfold speedup compared with a single-node server system, but the experiments nearly doubled the CPU time.

A significant mismatch exists between modern scale-out workloads and existing server processor designs. Modern server processor architectures use only a fraction of their available internal and external memory bandwidth when executing such tasks. To overcome this inefficiency, two main trends have emerged in recent years. The first is the use of many simple parallel-processing cores, either at the chip level or at the node level in the form of microservers. A typical scale-out workload executes simple instructions profiting from small and efficient cores, while many cores operate independently as there is little or no data dependency. The second trend is the use of specialized and heterogeneous architectures, such as system-on-chip processors in mobile devices or network processors in the telecommunication industry. These architectures have custom instruction sets or include dedicated accelerators tailored to an application domain.

Dedicated hardware accelerators can yield high performance and efficiency gains, but often lack flexibility when different or new tasks must be executed. On the one hand, a text-analytics query remains unchanged for a long time and operates on large volumes of data. On the other hand, the query is hand-crafted by a domain expert and can become very complex. A fixed architecture might not be flexible enough to execute new and complex queries. Thus, the text-analytics system must provide the flexibility of processing arbitrary text-analytics queries while identifying and accelerating bottleneck operations to
Several research groups have explored the use of hardware acceler- ators for efficient query processing. Such approaches have also been incorporated into commercial appliances. In one of the earliest exam- ples of such an approach, Kung and Lehman described systolic array- based accelerators for relational algebra operations.1 More recently, Mueller et al. proposed a query compiler that produces field-programmable gate array (FPGA) bitstreams for complex event-detection queries that consist mainly of relational algebra operations.2

Dennl et al. propose a system that enables on-the-fly composition of FPGA-based SQL query accelerators by combining a static stream-based communication interface and partially relocatable module libra- ries on the FPGA.3 Such an approach enables creation of FPGA bit- streams for dynamically changing relational queries without going through time-consuming FPGA synthesis tools. Sukhwani et al. describe an FPGA-based accelerator engine for database processing that offers a software-programmed interface to eliminate the need for FPGA reconfiguration.4 Chung et al. present a query compiler for the domain-specific Language Integrated Query (LINQ) language that can be mapped to accelerator templates.5 Wu et al. describe a programmable hardware accelerator for range partitioning that is directly attached to a CPU core.6 The accelerator operates in a streaming fashion, but accelerates only the range-partitioning step of query processing.

Our approach is inspired by IBM’s PureData System,7 which attaches FPGAs directly to storage devices to deal with large volumes of data. Although our accelerator architecture uses a shared-memory setup, a direct I/O attachment can be beneficial for specific use cases—for example, when documents are read from a database. To the best of our knowledge, our work is the first to produce FPGA-based accelerators that support a combination of information extraction operations (that is, regular expressions) and relational alge- bra operations.

A reconfigurable accelerator

Our system improves information extraction throughput by executing selected opera- tors on a reconfigurable device, such as an FPGA. One advantage of a reconfigurable device is that once it has been configured, it does not require any instructions to execute its tasks. The only data that must be transferred between the memories and the FPGA is the actual data to be processed, together with some negligible control information. In the case of text-analytics applications, which are typically applied to large volumes of data,
the same query is run for several hours or several days. Thus, fast reconfiguration capability is not needed.

Another advantage of FPGAs is their ability to compute in space. On the one hand, a reconfigurable device can implement a deep custom pipeline working on different data sets at different stages. On the other hand, multiple parallel instances can operate simultaneously on the same dataset executing different tasks, such as our architectures for the extraction operators.11,12 This high degree of parallelism makes up for the comparably low clock frequencies FPGAs provide. By moving not only single operators to the FPGA but also larger subgraphs of the operator graph, the parallelism can be fully exploited and the amount of communication between the software-based operators and the hardware-accelerated operators can be minimized.

In our framework, an operator graph (see Figure 1a) can be partitioned into both a supergraph (Figure 1b) and a hardware-accelerated subgraph (Figure 1c). Operators that are moved to the accelerator are removed from the original operator graph and replaced with a new subgraph operator. It is also possible to extract multiple independent subgraphs that can be executed in parallel or in sequence on the FPGA for the same or a different set of text documents. In this way, most of the unnecessary data gets filtered out before reaching the software modules running on the server processors, which greatly improves the processing rates. In this work, we have used the concept of maximal convex subgraphs13 to identify the subgraphs that are maximal in size and can be atomically executed without processor intervention.

To automate the generation of query-specific accelerators, we extended SystemT’s compilation flow. Figure 2 shows the acceleration flow added to the original SystemT text-analytics system. The information to be extracted can be formulated as an AQL query, which is then compiled into an operator graph, which is further processed by the original SystemT optimizer. Before deploying the operator graph, we perform a partitioning step that generates the software supergraph and the subgraphs that are run on the FPGA. We also developed a query compiler14 that uses a set of configurable operator modules that can be linked using an elastic interface to generate a streaming hardware design for a given subgraph.

The document is processed on the FPGA as a sequence of ASCII characters and is the only variable-length data structure used. The main data structure is a so-called span that defines a segment within the document text. A span is composed of a start and an end
offset, both of which are represented as 32-bit integers. Additional data types are integers, floats, and Boolean. The same type of operator can have different types of input schemas consisting of different numbers and types of data. However, all of these schemas are known at compile time, and our compiler generates a custom operator for each node in the operator graph.

The compiler leverages the possibility of implementing a large set of operators in streaming fashion when the input data is sorted in a certain direction. Sorting is itself a blocking operation, but many operators produce sorted or nearly sorted output data naturally. By adding simple sorting buffers or configuring preceding operators properly, the compiler ensures the accelerator’s streaming operation. After the compiler generates the hardware description, it is synthesized and the configuration is loaded onto the FPGA. The supergraph will be executed by the SystemT runtime on the host CPU, whereas the subgraphs are run on the reconfigurable accelerator.

The SystemT runtime uses multiple worker threads, all of which execute the same supergraph on different documents. When a worker thread reaches a subgraph operator, it notifies a dedicated communication thread, which coordinates the data transfers between the runtime and the FPGA (see Figure 3). Because of the document-per-thread execution model, we set the worker thread to sleep while the subgraph is being executed on the accelerator. To keep the CPU cores from idling, numerous worker threads run in parallel to hide the FPGA’s execution time. Ideally, the reconfigurable device would have a very low latency when accessing the data after receiving the instruction to execute its configured subgraph. However, traditionally, FPGA accelerators are attached via the system bus and access the data via direct memory access (DMA) transfers, which have a memory-access latency that is at least three to four times higher than the processor itself.15

Our accelerators use the load-store units of an early version of the coherent accelerator processor interface (CAPI).16 A service layer implemented on the FPGA enables the accelerators to access the processor’s main memory and operate in a common virtual address space with the applications running on the processor. In the system we use, the address translation is software-based and occurs within our communication thread, resulting in an additional communication overhead. To minimize the impact of this overhead, larger data blocks (greater than 1,000 bytes) should be transferred at once to fully use the system bus bandwidth. The communication thread can then collect the data submitted by some of the worker threads and generate a larger combined work package. It then sends that data to the accelerator’s work queue and

Figure 2. Extending SystemT’s compilation flow to support hardware accelerators based on field-programmable gate arrays (FPGAs). An additional partitioning step selects subgraphs that can be executed in hardware and generates the supergraph that runs in software. The hardware query compiler generates a Verilog description of the subgraphs that are synthesized for the FPGA.

Existing SystemT

Acceleration flow

IQL query

Original SystemT optimizer

SystemT runtime

Partitioning

SW supergraph

HW subgraphs

Compiler generating verilog

FPGA

Figure 2. Extending SystemT’s compilation flow to support hardware accelerators based on field-programmable gate arrays (FPGAs). An additional partitioning step selects subgraphs that can be executed in hardware and generates the supergraph that runs in software. The hardware query compiler generates a Verilog description of the subgraphs that are synthesized for the FPGA.
starts again to check for submissions from the worker threads. When the FPGA finishes working on a work package, it signals it via a status register to the communication thread, which wakes up the software threads that belong to this work package.

Experiments

We carried out several performance and profiling experiments on an IBM Power7 server running at 3.55 GHz, capable of 64 logical threads and 64 Gbytes of double-data-rate level 3 (DDR3) memory. We synthesized the accelerator designs for an Altera Stratix IV FPGA running at 250 MHz. The FPGA was attached to a proprietary bus interface capable of 2.5 Gbytes-per-second (GBps) DMA transfers.

Software measurements

We evaluated five customer queries that we ran over the same set of input documents. The SystemT profiler captures the time spent at each operator and accumulates it over the total runtime. From these numbers, we derived a relative distribution to get comparable profiles of our test cases (see Figure 4). Queries T1 to T4 are dominated by the processing time spent on extraction operators (RegularExpression and Dictionaries), whereas query T5 spends more than 80 percent on relational operators.

Extraction primitives operate across the entire document, whereas relational operators usually work on the results produced by extractors. The extraction operators are typically the slowest operations in software. As a result, the throughput for test case T5 is higher than for T1 through T4. Figure 5 shows the system throughput for all test cases running with different numbers of threads. Initially, the throughput scales nearly linearly with the number of threads before starting to roll off at eight. Surprisingly, the throughput strongly increases again between 32 and 40 worker threads. This behavior appears to be due to the operating system scheduler, which uses all logical threads on one processor before spawning to another one.

Accelerator measurements

As the query profiles show, a significant amount of time is spent on extraction operators that operate on all of the document text.
data. As a result, we optimized our hardware/software interface for this type of input so that the extraction operators on the FPGA determine the maximum achievable throughput rate regardless of the subgraph configured on the accelerator. A significant back pressure from the relational operators was never observed in our test cases, but could be removed by using shallow buffers at critical stages. In our experiments, we measured the throughput rate for an accelerator with four parallel streams and a maximum peak bandwidth of 500 Mbytes per second (MBps).

Figure 6 shows the measured throughput rate for different document sizes, which are submitted by parallel SystemT worker threads to the interface. We achieve the peak bandwidth with document sizes of 2 Kbytes or larger. News entries typically have a few kilobytes of text, and thus can be processed at the accelerator’s peak bandwidth. In contrast, when using 128-byte documents, the throughput diminishes by a factor of 10, and when using 256-byte documents, the throughput diminishes by a factor of five, even though the communication thread combines small documents into a larger work package. These numbers do not represent the size of a typical text document, but they do represent the typical size of Twitter messages and RSS feeds.

Figure 4. Relative time spent executing different operators for five real-life text-analytics queries. Profiling runs capture the time spent at each operator node of a query for a set of reference documents. Queries T1 to T4 are dominated by extraction operations, whereas query T5 requires more time on relational operators.

Figure 5. Throughput of the original software versus the number of threads for 256-byte documents. Measurements were run on a two-socket Power7 server capable of up to 64 logical threads. The performance gain between 32 and 40 threads appears to be a behavior of the operating system scheduler by utilizing the second CPU in the system.

**Analysis**

Our existing implementation of the SystemT runtime cannot execute the generated supergraph indicated by the dashed line in Figure 2. Therefore, we estimate the achievable overall system bandwidth by analyzing the results from the “Experiments” section. We observe that the runtime of most queries is dominated by extraction-type operators consuming up to 82 percent of
the overall runtime. Because all of these operators work on the same document data source, they are ideal candidates for acceleration on the reconfigurable device, where they can operate in parallel on a single document pass. Additional relational operators that are supported for hardware processing can add up to 97 percent of the total runtime.

The software throughput rate varies with the query's profile, whereas the hardware throughput is determined by the input operator of the subgraph. We choose to always offload the extraction operators, which allowed us to focus on the document data transfers. The document size has a significant impact on the accelerator's throughput, as Figure 6 shows. Although the peak bandwidth can be reached only by using larger documents, the throughput rate for smaller documents is still much higher than that of the pure software.

We estimate the overall system throughput using Equation 1, in which we add the remaining time spent on software processing, \( r_{SW} \), to the time spent on the accelerator using the measured throughput rates \( t_{SW} \) and \( t_{HW} \). The interface cost is included in our measurements for the accelerator throughput and does not need to be added as an extra penalty. We estimate the throughput achieved by offloading

- only the extraction operations to the FPGA,
- a single maximal convex subgraph that contains all extraction operations and as many hardware-supported operators as possible, and
- all hardware-supported operators to the FPGA using multiple maximal convex subgraphs.

In the first two cases, the estimations we present are pessimistic because we do not consider potential processing overlaps between the FPGA and the CPU. In the third case, our estimations are optimistic because we do not consider the communication overhead incurred by the additional subgraphs. Figure 7 shows our estimations when using 64 software threads, four hardware streams, and average document sizes of 256 or 2,048 bytes.
\[ t_{\text{out}} = \frac{1}{\frac{1}{t_{\text{HW}}} + \frac{r_{\text{SW}}}{t_{\text{SW}}}} \] (1)

Although the throughput rates of queries T1 through T4 increase up to 4.8 times by offloading the extraction operators, query T5 sees a limited impact. Only by running multiple subgraphs on the accelerator does query T5 gain up to threefold improvement. By offloading multiple subgraphs to the accelerator, query T1 improves by a factor of 10 for small documents and by a factor of 16 for larger documents.

As we enter the big data era, deriving value from large amounts of data efficiently becomes a necessity. We believe that text analytics will be a key application of this new era, but that it is challenged by the growing complexity of the queries and ever-more data to process. We have presented a prototype system that includes an FPGA as a reconfigurable accelerator and a hardware compiler that enables offloading selected parts of a given text-analytics query. Projections based on profiling results and actual measurements on the FPGA-attached system promise speedups of up to 16 times that of purely software-based solutions.

The speedup results reported in this article can be further improved by including support for additional relational operators in our hardware compiler. Further optimizations to the interface are also being investigated to minimize the latency penalty of small documents. Our future work will cover hardware/software partitioning algorithms to maximize the overall system’s throughput rate under the FPGA’s resource constraints. We also plan to identify the most power-efficient design choices for a given query. 

References


Raphael Polig is a predoctoral researcher in the Zurich Research Laboratory of IBM Research. His research focuses on reconfigurable architectures for text-based analytics. Polig has a Dipl-Ing in electrical engineering and information technology from the Technische Universität München.

Kubilay Atasu is a research staff member in the Zurich Research Laboratory of IBM Research, where he works on hardware accelerators. His research focuses on the design of reconfigurable hardware accelerators for big data and analytics applications. Atasu has a PhD in computer engineering from Boğaziçi University, Turkey.

Laura Chiticariu is a research staff member in the Infrastructure for Intelligent Information Systems group at IBM Research, where she works on building development support for information extraction systems, using a range of techniques including data provenance, information integration, and natural language processing. Chiticariu has a PhD in computer science from the University of California, Santa Cruz.

Christoph Hagleitner is a research staff member at IBM Research, where he manages the Accelerator Technologies group. His research focuses on performance- and energy-optimized hardware accelerators for next-generation computing systems. Hagleitner has a PhD in electrical engineering from the Swiss Federal Institute of Technology.

H. Peter Hofstee is a research staff member at IBM Research. His research focuses on heterogeneous and workload-optimized systems based on the Power architecture. Hofstee has a PhD in computer science from the California Institute of Technology. He is a member of IEEE, the ACM, and the IBM Academy of Technology.

Frederick R. Reiss is a research staff member at IBM Research. His research interests include information extraction, lossy and lossless data compression, high-performance query processing, streaming query processing, and query optimization. Reiss has a PhD in computer science from the University of California, Berkeley.

Huaiyu Zhu is a senior engineer at IBM Research. His research interests include machine learning, text analytics, and modes of computation. Zhu has a PhD in statistics and computational mathematics from the University of Liverpool.

Eva Sitaridi is a PhD candidate in the Computer Science Department of Columbia University. Her research focuses on database processing using GPUs. Sitaridi has an MSc in computer science from the University of Athens.

Direct questions and comments about this article to Raphael Polig, IBM Research-Zurich, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland; pol@zurich.ibm.com.