Xpriori Technical Paper

Three-Dimensional Content Scanning Using DPP™

by Chris Brandin

Release 1.1

NOTE: In October 2003, Xpriori, LLC acquired NeoCore Holdings, LLC including all technology and patents. Any references to Neo, NeoCore or NeoCore Holdings, LLC technology or patents as such are now the property of Xpriori, LLC.

Xpriori, LLC
2864 S. Circle Dr.
Ste. 401
Colorado Springs, CO 80906
(719) 425-9840
www.xpriori.com

Introduction

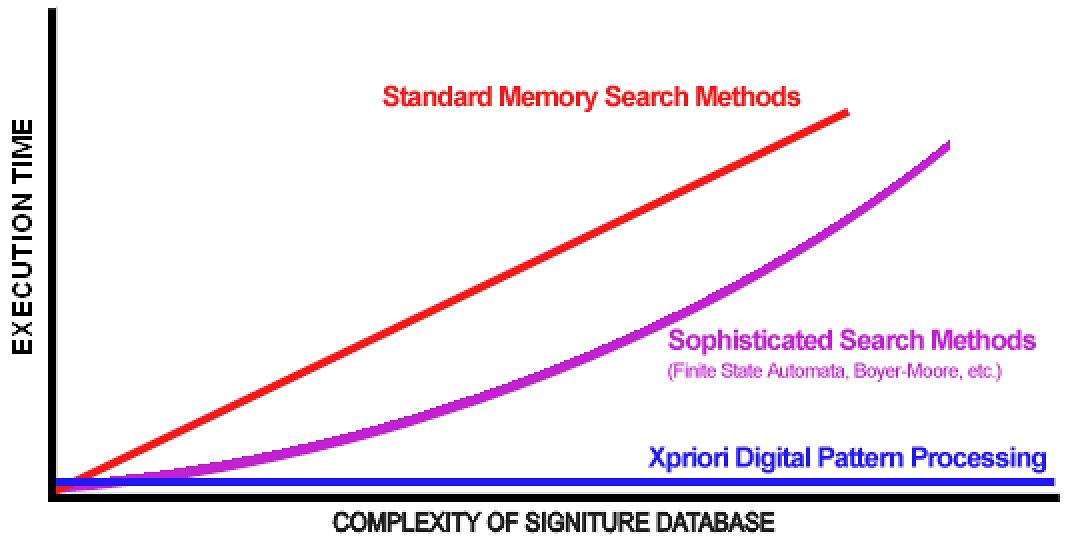
One application for which Digital Pattern Processing™ (DPP) is particularly well suited is three-dimensional content scanning. Scanning for instances of a single signature (like looking for a word in a text file) constitutes a one-dimensional search – there are many good algorithms for doing this. A two-dimensional search consists of looking for multiple signatures of a single width (there are several suitable methods for accomplishing this as well). Three-dimensional content scanning involves searching for multiple signatures of varying widths – this can be a daunting task. Current methods of accomplishing this do not scale well; that is, as the number of signatures and the number of different signature widths increases, performance degrades correspondingly. Using DPP technology, a three-dimensional content scanning engine can be built that is largely oblivious to the number of signatures (and their various lengths) being sought.

The Problem

If we examine the various methods employed to accomplish three-dimensional signature scanning a number of issues arise. If naive methods are employed, the following process will generally govern performance:

1. Create one hash-table for each different signature width.
2. Start at the beginning of the data block to be scanned.
3. Produce a hash-code for the first signature width.
4. Attempt a hash-table lookup
5. Produce a hash-code for the next signature width.
6. Attempt a hash-table lookup.
7. Repeat steps 5-6 until all signature widths have been exhausted.
8. Move over one byte.
9. Repeat steps 3-8 until all bytes in the data block have been exhausted.

If we were searching for signatures of 24 different widths in a 60M document, over 1.4 billion hash-table operations would be required – and that would take a very long time. More sophisticated methods of accomplishing this can be employed (FSM's, variations of boyer-moore and karp-rabin, etc.) that optimize this process somewhat, but they eventually fail as well. The following diagram compares the performance of various methods for three-dimensional content scanning, including DPP.



Note that the performance for DPP is essentially flat. In DPP performance tests, the difference between scanning for a few signatures of the same width, and scanning for huge signature lists representing a large variety of widths, was barely measurable. On a 900mhz Pentium III computer, data blocks were scanned at the rate of 100 mega-bits per second (fast Ethernet rates) using a software implementation of the DPP-based three-dimensional scanning method. The benchmark consisted of looking for 20,000 signatures of 24 different sizes in a 60M document, resulting in 500,000 hits. This is equivalent to doing a lookup against the entire signature database (all signatures of all widths) every 120ns.

DPP Applied to Three-dimensional Content Scanning

All elements of DPP are used in three-dimensional content scanning applications: virtual associative memory, symbolic processing, and simplified versions of field descriptors and behavioral sets.

To illustrate the methods employed we will use a simple example. Documents are to be scanned for all instances of several words. The words are of varying lengths; some are six letters, some seven and eight. First, a signature Store (database) of these words is created in the virtual associative memory. The shortest signature width is six letters, so this will become the “base set”. For each signature word an entry will be created that is an “association behavior” Quanta (one with an association) – basically that means that it returns an association when matched. For the seven and eight letter words, additional entries will be created based on the first six letters of each word (they can all be put into the same Store). These entries will be “qualifier behavior” Quantas - they indicate that there may be a “hit”, but additional letters must be examined as specified by the appropriate “Field Descriptors”. Because associations are unnecessary in “qualifier behavior” Quantas (at least for this application), the association field is used to indicate how many additional letters need to be examined in lieu of using a separate Field Descriptor. So, what we now have is a DPP engine that supports two behavioral set types (association and qualifier), and a simplified form of Field Descriptors. DPP accomplishes three-dimensional content scanning as follows:

1. Create a signature Store as described above.

2. Start at the beginning of the data block to be scanned.

3. Iconize the first six letters (the number of bytes in the base-set). This is the starting value for the "base set Icon".

4. Attempt a lookup against the signature Store. One of three things will happen: nothing – in which case skip to step 8, an association is found (there is a "hit") – in which case service the hit and skip to step 7, or a "qualifier behavior" Quanta is found. If a "qualifier behavior" Quanta is found:

    5. Create a new Icon by continuing the Iconization process to include the additional number of letters specified by the Field Descriptor (in this case, the association field value).

    6. Return to step 4.

7. Use Icon Algebra to cut the first letter of the current "window" into the data block from the base set Icon and continue Iconizing with the next letter after the window. Basically, this amounts to "sliding" across the data block.

8. Repeat steps 4-7 until all letters in the data block have been exhausted.

The method described above has a number of advantageous characteristics. First, the width of the base set signatures has no effect on performance – because Icon Algebra is used to only process the letter being removed and the one being added. This means that only two operations are executed irrespective of the number of letters in the base set signatures. These two operations are collapsed into one when the IG/APU is implemented as hardware. As long as the base set signature width is at least three letters, "qualifier behavior" Quanta operations are so infrequent that their effect on performance is negligible. DPP based virtual associative memories require an average of 1½ memory cycles for each lookup, irrespective of the number of entries accommodated – so the number of signatures in the Store have no effect on performance. DPP Stores also use memory deterministically (they have no overflow areas and they can be filled to 100% capacity), and signatures are represented by fixed-field symbols (so the length of signatures has no effect on store size). Icon size can be tuned to render the possibility of "false positives" either entirely irrelevant or altogether impossible.

Additional Features for Specialized Applications

Adding additional features to the content scanning method described above can further optimize specialized applications. To illustrate this, one such application will be examined – URL filtering. Two assumptions can be made about this application: network packet payloads will have to be scanned that may be fragmented, and signatures may contain a mix of upper-case and lower-case letters as well as hexadecimal representations and extra white space characters.

To understand how the first issue (fragmented data blocks) can be dealt with one has to understand a little about how modern microprocessors work. Today's pipelined super-scalar microprocessors (that would include virtually all current models) are actually tightly knit multi-processors; that is, they contain several processors running in parallel. It would be natural to assume that the more instructions being executed, the more time it takes. This is not necessarily true. A microprocessor with four execution pipes (four processors) will be capable of executing four instructions simultaneously as long as there are no data-dependencies between those instructions. When there are data-dependencies pipes are “stalled” – in other words, they don’t do anything until the data-dependency has been resolved. Typically, microprocessor pipes end up spending a lot of time waiting because no non-dependent instructions are waiting for execution. The additional instructions necessary to support a series of data block fragments in a way that they are treated as one big block, create few (if any) data dependencies with the primary DPP operations necessary to search for signatures. The result is that the addition of those instructions has little impact on performance (typically less than 10%).

The second issue (upper-case vs. lower-case vs. hex-represented vs. white-space characters) can be accommodated by pre-parsing the data block input stream before Iconization. This technique is similar to one commonly used for search engines. For example, let us suppose that we want to filter access to a URL called “http://www.pornographic-page.com”. That one URL could be entered as follows (and this list is by no means complete – the possibilities are endless):

http://////////www.pornographic-page.com
http://www.Pornographic-page.com
http://www.pOrnographic-page.com
http://www.pornogr%41phic-p0x41.com
http:////www.pOrNoGrApHiC-pAgE.com
http://www.PORNOGRAPHIC-PAGE.com

First, all signatures are converted to lower-case, all white space characters are converted to a common character (a “space” will do), and extra white space characters are skipped. The data block stream pre-parser contains two elements: a 256 entry translation table representing all possible 8-bit characters, and a state machine employed to skip extra white-space characters and process hexadecimal sequences. When a byte of input data is read it is used to address the translation table, and the corresponding table entry is presented to the Icon Generator in its place. The table causes all non-white space characters to be converted to lower-case and all white space characters to be converted to the same character (typically a “space”). Also, the table contains flags for each entry indicating whether it represents white space or if a possible hexadecimal sequence has started (or should be stopped or translated). When the white space flag is encountered, the state-machine is started causing all input to the Icon Generator to be blocked until a non-white space character is encountered, then it will be stopped and input will be allowed to the Icon Generator. In the case of hexadecimal sequences, the state machine is activated to manage that circumstance. All entries in the example list above would be converted to a common representation: “http:/www.pornographic page.com”. The translation table is programmable, so other parsing schemes can be implemented. The effect of this scheme on performance is minimal - for the same reason cited in the prior paragraph. Also, because redundant white space is never passed to the Iconization process, skipping characters is as fast as Iconizing them.

The DPP method for content scanning can be applied to specialized hardware implementations as well, resulting in totally scalable performance. Input data blocks can (obviously) be split-up and processed in parallel. DPP tasks can also be “pipelined”; for example, base set processing can be done by one processor, while “qualifier behavior” processing can take place on another. A number of parallelism and pipelining techniques can be applied to DPP, on several levels, in order to achieve most any performance level.

Conclusion

Digital Pattern Processing is very well suited to three-dimensional content scanning applications, offering superlative performance that remains flat irrespective of signature database complexity. This application serves as a good introduction to the basic principles underlying Behavioral Sets and Field Descriptors, and how they relate to each other. Also, an example is shown illustrating how other methods of processing can be incorporated into a DPP to achieve a higher-level function.

Appendix

Related Papers

Brandin, Chris, “A Definition of Digital pattern Processing™”, NeoCore, 2000

Brandin, Chris, “DPP™ Memory Management”, NeoCore, 2000

Brandin, Chris, “Optimized Coding Methods for Icon Generation and Manipulation In DPP™”, NeoCore, 2000

Brandin, Chris, “Management of Duplicate Data Elements in DPP™ Virtual Associative Memories”, NeoCore, 2000

Brandin, Chris, “Behavioral Set and Field Descriptor Implementations in DPP™”, NeoCore, 2000

Direen, Harry and Phillips, Keith, “Finite Fields and Properties of the NeoCore Icon Generator, Associative Processing Unit, and Associative Memory Controller used in Digital Pattern Processing™”, NeoCore, 2000