Xpriori Technical Paper

A Definition of Digital Pattern Processing (DPP™)

by Chris Brandin

Release 1.1

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

Version 1.1

Xpriori technology is protected by the following patents:
US Patent #5,742,611 (21 Apr 98)
US Patent #5,942,002 (8 Aug 99)
US Patent #6,157,617 (5 Dec 00)
US Patent #6,167,400 (26 Dec 00)
US Patent #6,324,636 (27 Nov 01)
US Patent #6,493,813 (10 Dec 02)
US Patent #6,792,428 (14 Sept 04)

Other U.S. and international patents pending.

The information in this white paper has been provided by Xpriori, LLC. To the best knowledge of Xpriori, it contains information concerning the current state of information processing technology. Xpriori, LLC disclaims any and all liabilities for and makes no warranties, expressed or implied, with respect to products described in this paper, including, without limitation, the implied warranties of merchantability and fitness for a particular purpose. No specific reliance should be made on the material provided herein without thorough investigation of the technology and its proposed application to specific circumstances. Product and technology information is subject to change without notice.

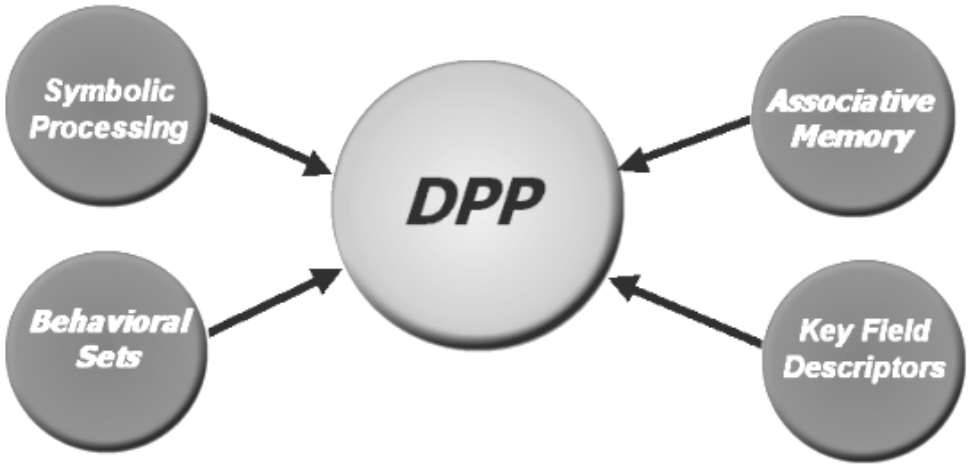
Introduction

Digital Pattern Processing (DPP) lends itself to a specialized type of information processing much like Digital Signal Processing (DSP) is targeted toward signal processing tasks. Pattern Processing is not simply a method for addressing pattern recognition applications; rather it is a completely new way of looking at and processing information and it can be applied to a wide variety of data processing challenges. Today DSPs are used in many applications that go far beyond the signal processing tasks for which they were originally intended. Similarly, DPPs can be applied to many tasks that bear little overt relationship to pattern recognition.

This paper addresses the lowest level components of Digital Pattern Processors. If we were discussing DSPs, this paper would be about multiply-accumulate functions and Harvard architectures. This paper discusses the lowest level components without which the more understandable functions of Pattern Processing would not be possible. The discussions are primarily architectural. Other papers (referenced in the Appendix) address issues ranging from formal mathematical characterizations to higher-level applications. Some of these papers approach applications in utterly unorthodox ways that would simply make no sense without the basic elements described here.

The Four Basic Components of Digital Pattern Processing

There are four basic elements that constitute Digital Pattern Processing; associative memory, symbolic processing, field descriptors, and behavioral sets. Each element has individual value, but when combined heretofore impractical processing tasks can be accomplished with ease. If any one element is left out, the power of DPP is significantly reduced because the components are designed to work together in an architecturally cohesive manner.

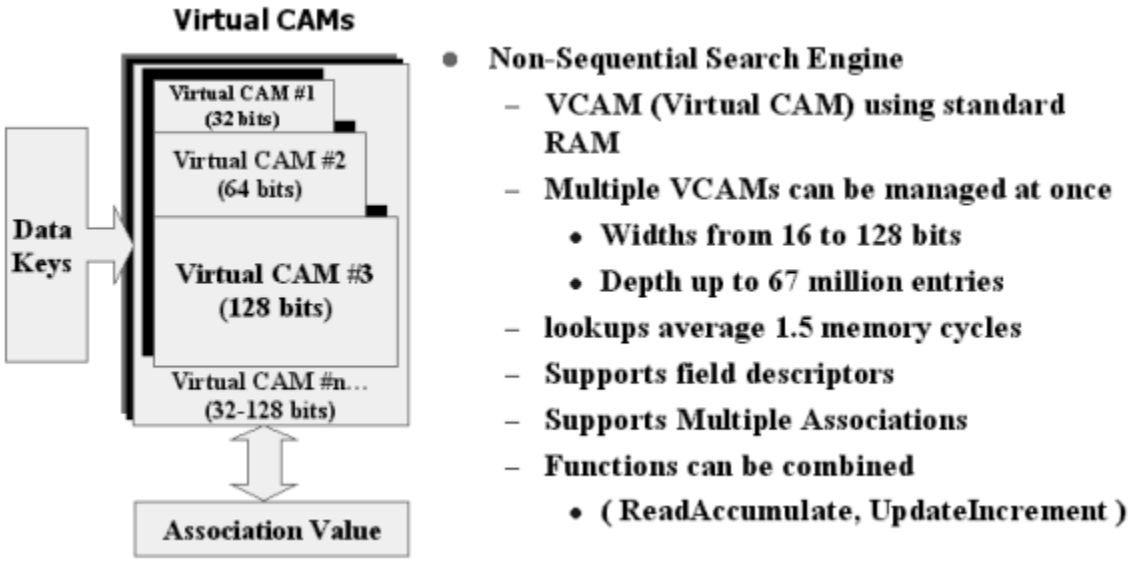


Associative Memory

Associative memory is a simple concept. Instead of responding to an address with memory contents at a particular location, associative memory recognizes a data pattern and returns a data element associated with that data pattern. It is almost the architectural inverse of conventional memory, except the association is not limited to being an address – it can be anything. Conventional memory requires that a processor know *where* information is whereas associative memories directly relate data to *what* they are presented with. This allows memory

to behave in a more natural way that relates more directly to what we do with computers; the result can be significantly improved performance for many tasks.

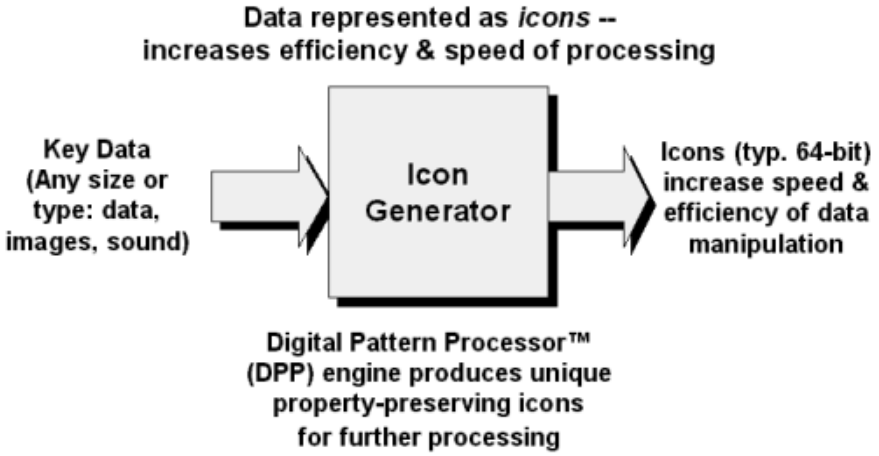
Associative memories have been around for a long time. Content Addressable Memories (CAMs) are a form of associative memory. In order to accommodate DPP a virtual binary associative memory on a much larger scale than previously available is required. DPPs must be able to allocate multiple associative memory pools of various configurations with no pre-determined restraints on what those configurations might be. Also, unusually wide (256 bits) and deep (billions of entries) configurations may be required. Xpriori associative memories are designed to be scalable to extreme configurations, size having no effect on performance whatsoever. They are also virtual; that is, multiple instances of memory pools can be simultaneously supported. The use of memory is both efficient and deterministic. If a 1M 64-bit CAM with a 32-bit association is required, for example, the memory required is 12M – which is the same amount required to simply contain the key-association pairs. In other words, the overhead required to organize the data is zero. The associative memories are housed in conventional RAM requiring an average of 1.5 memory cycles to find an entry. As RAM is faster than hardware based CAM, the virtual associative memories are faster than their hardware alternatives.



Symbolic Processing

The next element – Symbolic Processing – is a simple, yet curiously difficult concept to grasp. With virtual CAMs (as discussed above), there is a one-to-one mapping between memory and associations; that is, no information is lost. With Symbolic Processing, information is represented by symbols that retain logical properties of information without containing the original data – these symbolic representations are called "Icons". Icons are always fixed length – typically (but not necessarily) 64-bits. They "stand in" for information for the purposes of processing. To achieve this, information is bifurcated into two components: data and logical properties. An Icon is used by the DPP to locate an association. Basically Icons are fixed field representations of the logical properties of any block of data desired, irrespective of its length of type. Icons are linearly algebraic; they respond to logical processes the same way that key data would if it were changed and then "Iconized". For example, if "Albert" were iconized, it may produce a value of I35. Similarly, Iconizing " Einstein" and "Albert Einstein" may produce I46 and I31 respectively. Alternatively, we can combine I35 and I46 to produce I31 without needing the original data that produced the constituent Icons. Because Icons are computer-width words that can be processed in a single step, iterative logical manipulations so common to many processing tasks can be

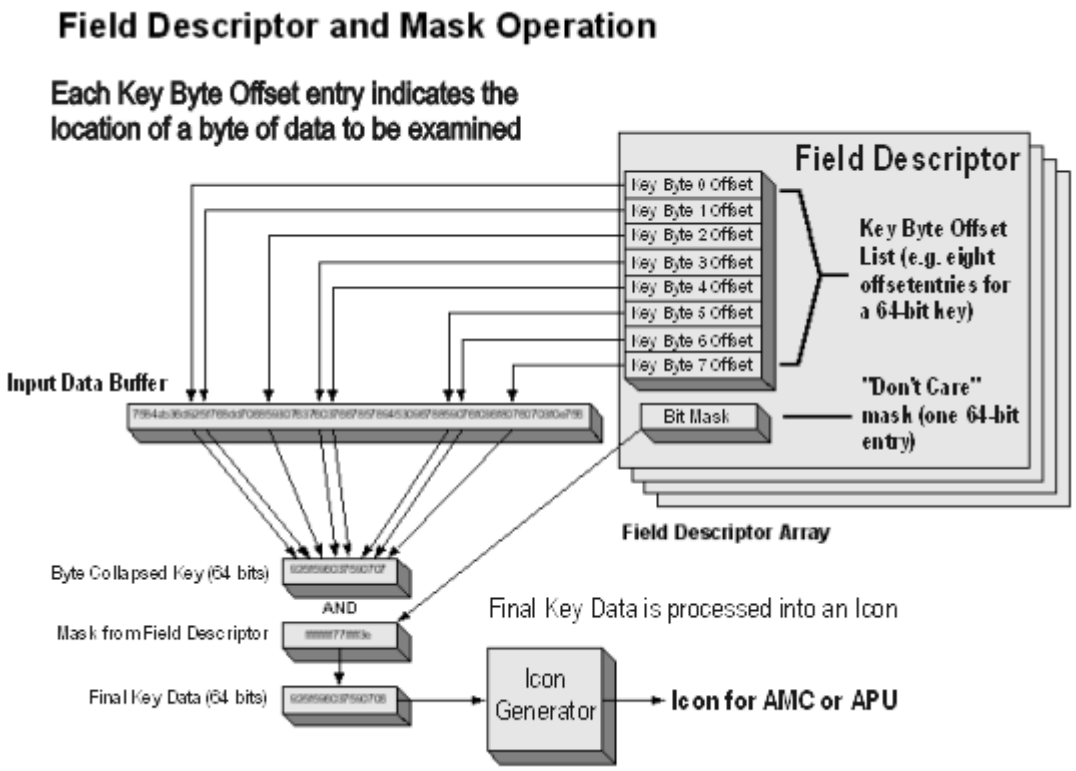
achieved arbitrarily faster than manipulating original key data and then re-processing the result. A variety of virtual logical processes can be exercised against Icons, including combining, removing, and changing information. This type of processing is called "Icon Algebra". As only the resulting Icons are used to locate associations, the processing task at hand can be achieved without the original key data.



Because "ragged" data is represented by fixed-length symbols there is a possibility that different blocks of data can produce the same Icon, resulting in "false positives" (or "collisions"). Happily, the odds against this are very high – Icons are generally more reliable than any machine you can use them on. In fact, because less information is processed, system reliability is actually increased. Using a 64-bit Icon the odds that any two bodies of data will collide, producing the same Icon, are approximately 18,000,000,000,000,000,000 to 1. One would be 500 times more likely to win the state lottery and be struck down by lightening in the same day. Even factoring in the "birthday paradox" with large data sets, collisions are extremely unlikely. A larger Icon can be used, 256-bit for example. The odds against a collision in that case are utterly astronomical – if someone were to select a single atom from our galaxy at random, and somebody else another, they would be two-billion times more likely to select *the same atom*. In many applications the possibility of collisions is inconsequential. For the others, the Icon can be "tuned" (at the cost of additional memory) to achieve any level of reliability desired – right up to eliminating the possibility of collisions altogether.

Field Descriptors

Field Descriptors are used to describe the location of key data so that the DPP can fetch it in lieu of being presented with a pre-processed pattern. This allows patterns with "don't care" elements to be fragmented over an arbitrarily long data block. Perhaps more significantly, they make the last element – Behavioral Sets – possible. A field descriptor consists of two elements: a stack of Key Byte Offsets and a "Don't Care" Mask. Each Key Byte Offset entry points to one byte of data to be processed for Iconization. Once the bytes have been gathered from a block of data, a bit-mask is applied in order to specify which bits within the bytes are to be excluded. This scheme was designed to be optimal for the processor/memory schemes employed by most computer systems.



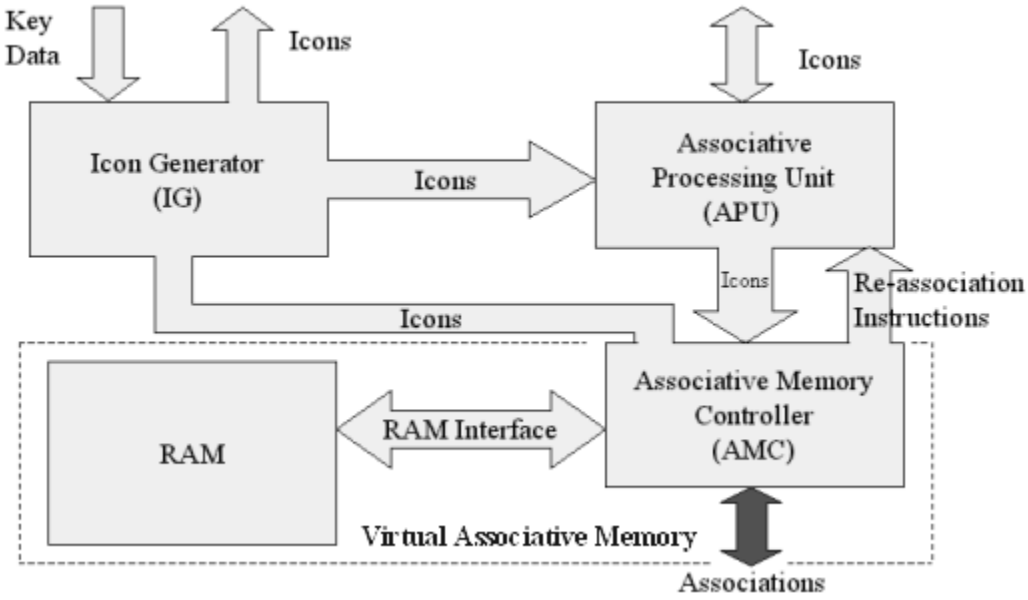
Behavioral Sets

The purpose of behavioral sets is to bring "predicate logic" processing capabilities to DPP. So far, the functions described allow one to efficiently retrieve associations in sophisticated ways. Although this functionality is valuable by itself, it is desirable to have the ability to make decisions during the association process. This requires one more feature to be supported. Behavioral Sets allow an association to contain additional instructions for the DPP. For example, in order to associate the range 00010000b – 00011110b, a couple of different strategies could be employed. One could simply create fifteen entries with the same association. With Behavioral Sets the same can be accomplished with two entries. The first entry associates the range 00010000b – 00011111b based on a Field Descriptor that treats the four least-significant bits as "don't cares" (11110000b – a "0" masks out the bit), and it contains a flag indicating that another association should be looked for based on a Field Descriptor specifying that all bits be examined (111111111b). An association will be returned for the fifteen entries we want and one invalid entry (00011111b). If the second look-up matches the 00011111b entry, a flag indicating that this is an exception to the previous association will cause that association to be abandoned. A variety of behaviors are accommodated, these include find exceptions, find exceptions to exceptions, look at additional fields based on what was found, search from smallest match (a tree), search from largest match (an upside-down tree), search in any order desired, match based on proximity, apply an offset to a pattern and search, etc. With Behavioral Sets "fuzzy" matching, annealing, and "drill-downs" can be accomplished with efficiency. Furthermore, because behaviors are not algorithms, different methods can be combined into one DPP store

where each individual association process is dynamically defined by the associations themselves. The DPP is guided by what it "knows" rather than being governed by a pre-determined process.

DPP Implementation

The basic architecture of a DPP remains the same whether implemented in hardware, software, or a combination of both. The Xpriori DPP consists of four components: the Icon Generator (IG), the Associative Processing Unit (APU), the Associative Memory Controller (AMC) and memory (RAM).



The IG converts key data into an Icon; that is, a finite-field representation of the logical properties of said key data. If the resulting Icon is as long or longer than the number of key data bits that can change and are not "don't cares", then the transform is perfect; no information is lost and there is no possibility of two key data items producing the same Icon. If the significant portions of key data add up to more than the Icon length, then the characteristics described in the "Symbolic Processing" section above apply. The coding method employed by the IG is both fractal and extensible, so a common IG unit can produce Icons of any desired length without having to be re-programmed. The IG converts arbitrary key data to a fixed-length, property-preserving Icon that is used for processing and association. The IG also fetches elements of key data as defined in Field Descriptors.

The APU is used to accomplish all "Icon Algebra" tasks. It also processes re-association instructions as described in the "Behavioral Sets" section above.

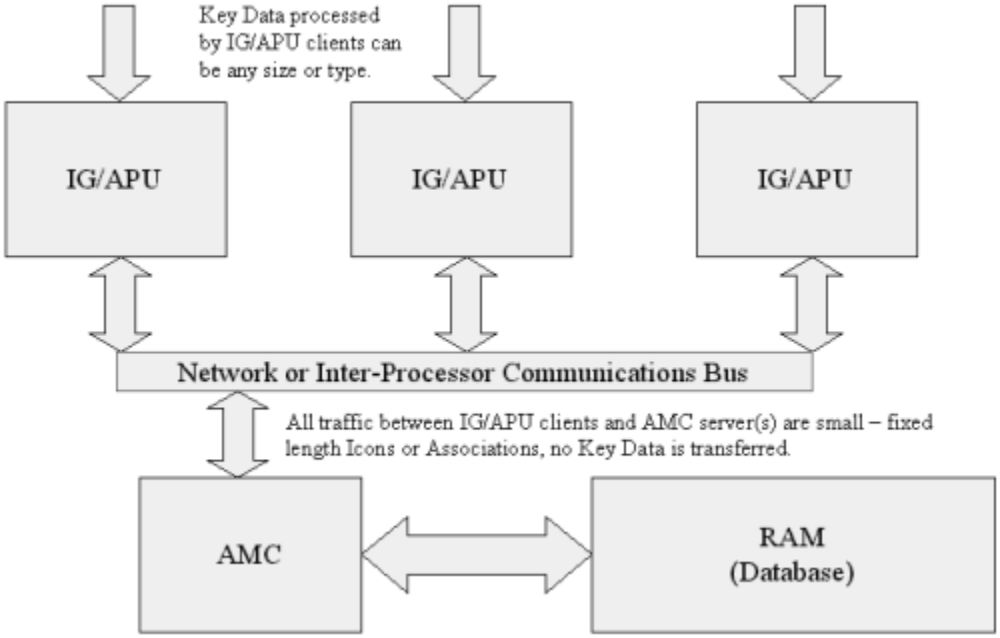
The AMC manages memory. Based on an Icon, it locates and returns an association in an average of 1.5 memory cycles.

All components can be implemented as hardware or software. In some high performance applications, the IG and APU are implemented as dedicated hardware and the AMC function is implemented using a RISC processor. An extremely high-performance device might consist of a

RISC processor coupled with an IG/APU core, much like a DSP core could be added to a RISC processor. The components of a DPP lend themselves to being pipelined as well; so even higher performance levels can be achieved using more esoteric architectural schemes.

Distributed Architectures

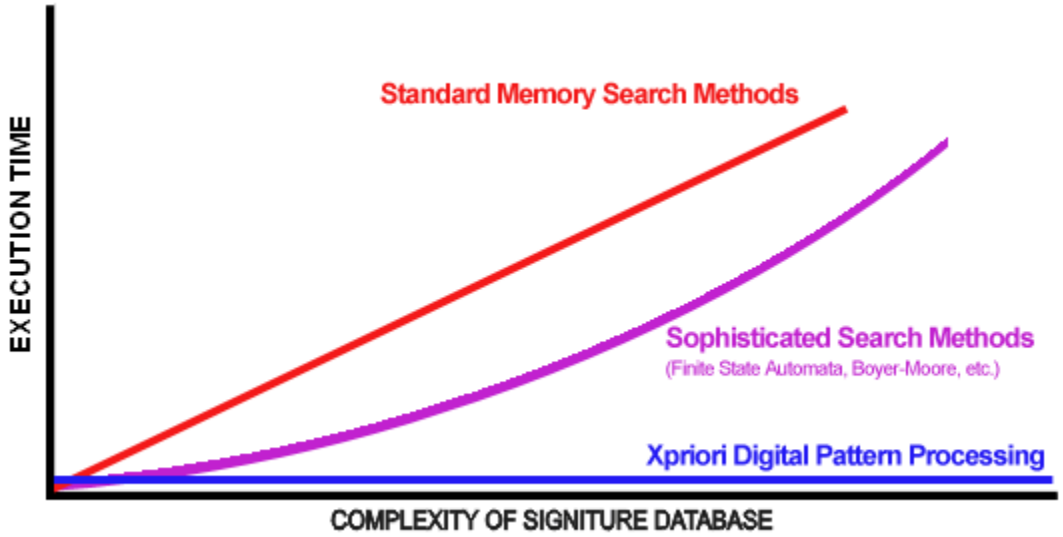
Upon examination of the DPP architecture it becomes apparent that the IG/APU functions and the AMC/RAM functions can be treated as discrete processes. This can be used to advantage in many applications. Pattern recognition applications, for example, that make heavy use of Icon Algebra can be implemented using parallel (or distributed) processors where IG/APU functions are executed on client processors sharing a common association database (AMC/RAM functions). Because "ragged" key data is processed into fixed-field Icons *before* it is passed to the AMC, inter-processor data exchange volume is held to a minimum. Also, mechanisms for locating and filtering searches for information can be separated from the data itself.



Conclusion

Implemented in software alone, DPPs can achieve orders-of-magnitude performance increases over conventional processors for a wide variety of processing tasks. Using hardware, even higher performance levels can be achieved. DPP was developed with a simple goal in mind – to be able to do well what computers have traditionally done badly. Rather than resorting to parallelism to incrementally improve performance for certain applications, we re-examined the meaning of information processing on a more basic level. Having seen hand-held calculators outperform any person in numerical tasks, and likewise having seen insects outperform the world's largest computers in pattern based tasks, it was easy to conclude that there was potential for a fundamentally new way to process information. DPP is a different way to deal with information, much like DSP is different way to process signals.

There are certain assumptions about the complexity of processing tasks that do not apply to DPP. For example, if one examines the methods currently employed to accomplish three-dimensional content scanning, it is natural to conclude that the more complex the search, the longer it takes to complete. Using DPP, this is not true. DPP can process one signature or millions of signatures of thousands of different sizes simultaneously, with no substantial difference in performance.



The diagram above shows a comparison of naïve vs. sophisticated three-dimensional search methods pitted against DPP. Performance is always related to the complexity of the task - except when DPP is used. DPP performance is not proportional, curved, or even linear – it's flat. This diagram can be applied to other tasks as well, such as pattern matching, fuzzy searching, data annealing, genome matching, storage and management of XML data, pattern recognition, data security, document protection, CAMs, and more. In some cases the "time of execution" axis can be replaced with "component count" or "cost".

DPP can be used to improve conventional applications. Even more significant are the new, heretofore unavailable, application spaces Digital Pattern Processing opens up.

Appendix

Related Papers

Brandin, Chris, “DPP™ Memory Management”

Brandin, Chris, “Three-Dimensional Content Scanning Using DPP™”

Brandin, Chris, “Optimized Coding Methods for Icon Generation and Manipulation in DPP™”

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

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

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

# # #