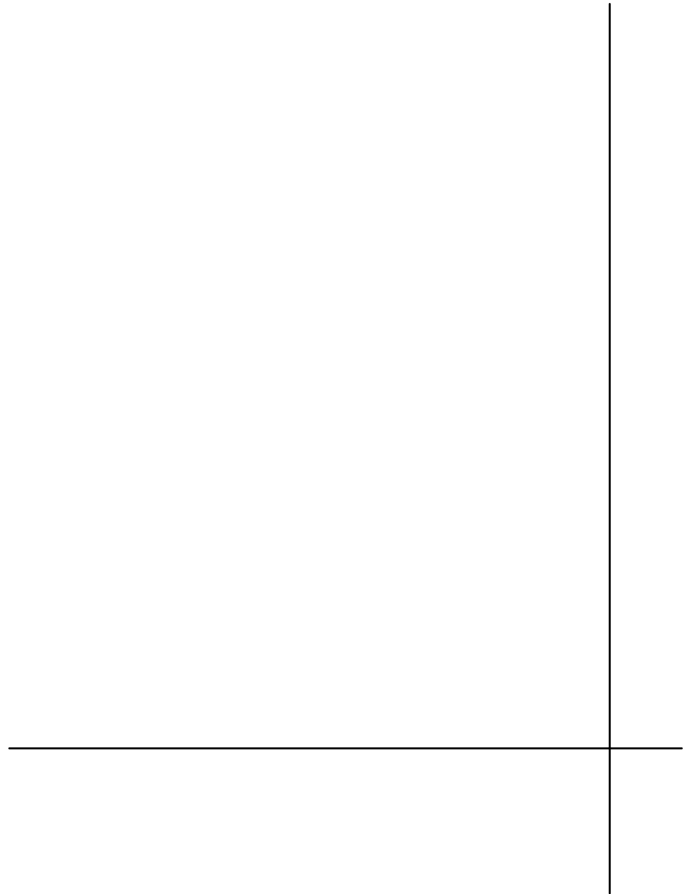
High-Performance Computing Algorithm Analysis

A Tarari White Paper



This page intentionally left blank

Executive Summary **4**
Overview of Tarari Solution **5**
Content Processing Controller 5
Content Processing Engine 6
Zero Bus Turnaround, Synchronous Static RAM (ZBT SSRAM) 9
Double Data Rate, Synchronous Dynamic RAM (DDR Memory) 10
Workflow on the Tarari Processor 10
Evaluating Applications for the Tarari Processor **12**
A—Algorithm Acceleration 12
B—Benefits of Offloading 16
C—Compatibility of Hardware Platform and Algorithm 19
D—Dynamically Reconfigurable Hardware 21
Algorithm Analysis Examples **23**
1. Biotechnology Research 23
2. Cryptography Acceleration Agent Set 24
Appendix A—Summary of Guidelines **28**
Appendix B—Advanced Topics **30**

Executive Summary

The Tarari® High-Performance Computing Processor accelerates the execution of complex algorithms used in high-performance computing (HPC) applications. The Processor allows high-performance computing users in industry, government and education to accelerate complex and compute-intensive applications in areas such as:

- Biotech
- Embedded HPC for signal processing
- Embedded HPC for image processing
- Communications
- Entertainment
- Seismic
- Financial/commercial

Tarari High Performance Computing Processors (or “Tarari Processors”) accelerate complex algorithms by moving the core algorithms off of traditional processors and into reconfigurable logic. The Tarari Processor is based on dynamically reconfigurable hardware, which means that multiple “agent” algorithms (code for reconfigurable logic) can be loaded or changed “on the fly” by user software on the host processor. Dynamic reconfigurability also allows for easy upgrades and further improvements in acceleration.

Using this document, system designers can determine whether their particular applications are good candidates for acceleration of certain algorithms on a Tarari Processor. The document ends with concrete examples of applying the Tarari Solution.

Overview of Tarari Solution

The Tarari Solution is:

- the Tarari Processor, based on an integrated processing platform (see Figure 1--Tarari Processing Platform Block Diagram);

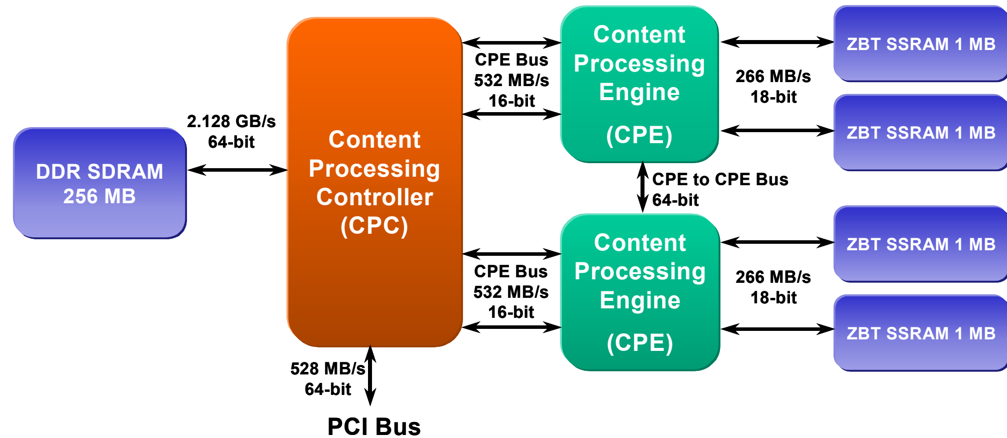


Figure 1--Tarari Processing Platform Block Diagram

- reconfigurable logic that can target specific compute-intensive tasks and decrease the processing time required to perform them.

Content Processing Controller

The Content Processing Controller is the heart of the Tarari Processor, connecting the PCI bus, DDR SDRAM ("DDR memory"), and the Content Processing Engines. It also contains the Configuration Logic, which initializes and dynamically reconfigures Acceleration Agent Sets that are loaded into the Engines. With four independent busses, the Content Processing Controller allows multiple algorithms to run simultaneously, and it allows data to move to and from the onboard DDR memory at high speed.

The importance of the Content Processing Controller is that it drives reconfigurable logic on the Tarari Processor. It can dynamically reconfigure **Acceleration Agent Sets** (groups of acceleration agents that solve particular problems and run in the Content Processing Engines) within 30 milliseconds so that they can solve different problems or even different parts of the same problems.

Reconfigurable logic:

- provides the ability to load new agents to partially or completely change functionality, *without requiring changes to the hardware*;

- can load new agents while existing agents continue processing;
- does *not* require a system re-boot.In the example shown

in Figure 2--Reconfigurable Logic, the Content Processing Controller reconfigures logic on the Tarari processor to change it from an SSL accelerator to an SSL/XML/GZIP accelerator. Note that some of the RSA capacity is reconfigured for XML parsing and the DES/3DES capacity is reconfigured for GZIP (green dashed outline), while RNG and some RSA capacity remain unchanged (red outline)*. The option of reconfiguring *all or part* of the logic—"partial reconfigurability"—is another feature of the Tarari Processor.

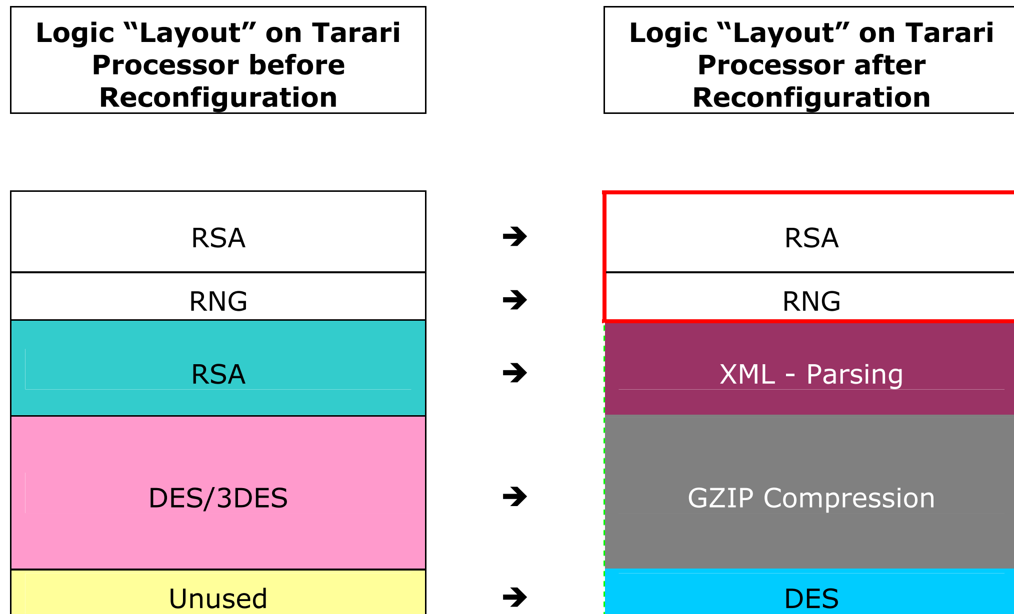


Figure 2--Reconfigurable Logic

Finally, the Content Processing Controller's capacity for managing traffic and processes such as agent-chaining (see below) on the Tarari Processor is what permits the Content Processing Engines to achieve algorithm acceleration and higher overall system performance, while insulating the designer from the details of bus management and intelligent data management.

Content Processing Engine

In the Tarari Processor are two Content Processing Engines. These Engines embody reconfigurable logic in the form of hardware gates, Block-RAM ("BRAM"), and specialized functions such as multipliers and clock managers. At system startup a software-based Agent Configuration Manager loads the

application-specific **Acceleration Agent Sets** into reconfigurable logic in the Engines to accelerate algorithms and applications. With the tremendous flexibility of the Engines, users enjoy high-performance computing benefits such as:

- **Pipelining**—Pipelining allows all instructions in an algorithm to execute on every clock cycle. Unlike a traditional processor (where the instructions are fetched, decoded and then executed), each element of the logic on a Tarari processor can be executed at the same time. This is similar to a "bucket brigade" which moves water from each person to the next, with each bucket moving at the same time. Each piece of logic "executes" at the same time.

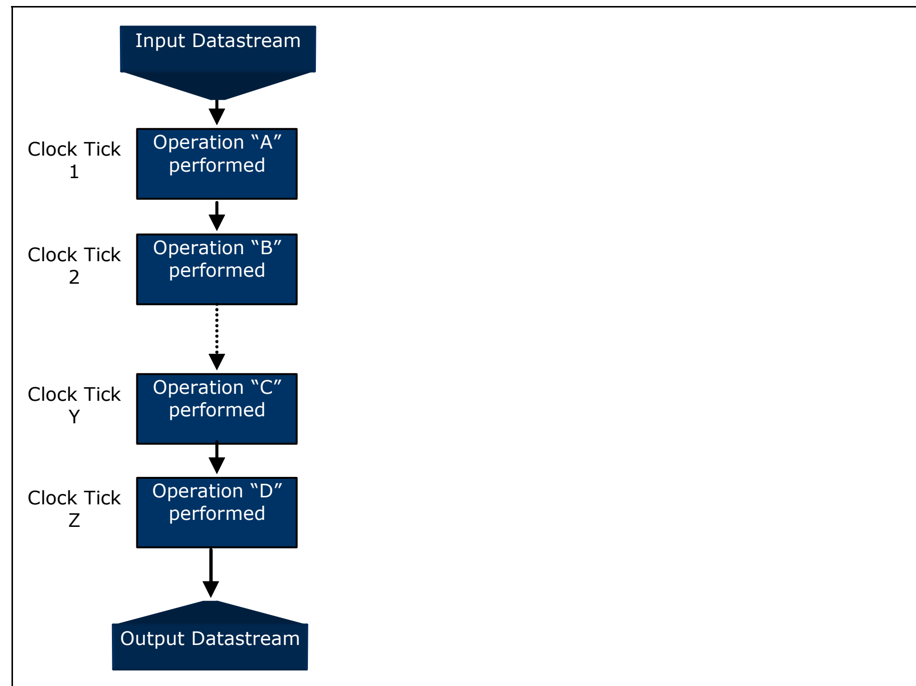


Figure 3--Pipelining

The ability to register intermediate results allows for a higher clock rate of logic design and higher throughput. This results in less idle capacity on the Tarari Processor and an increase in the number of instructions that can be performed during a given time period. Even if there is some sequential dependency, a pipelined application can take advantage of those operations that can proceed concurrently.

- **Parallelism**—It is possible for multiple instantiations of an agent or multiple agents to be executed in parallel. In

a regular expression matching application, the benefits of acceleration are multiplied

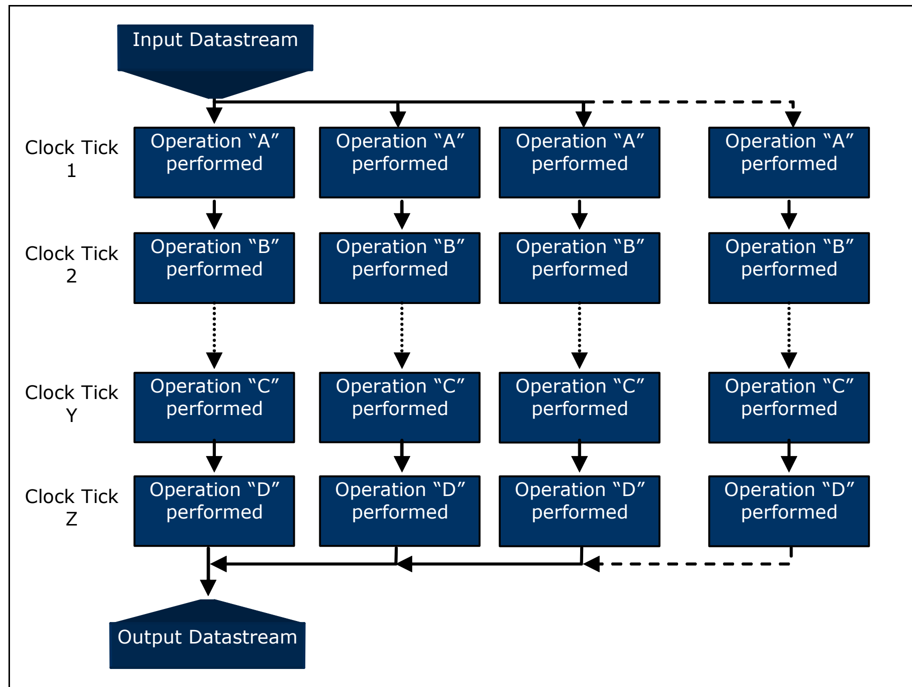


Figure 4--Parallelism

as more instantiations of the same agent are added to the Acceleration Agent Set. Five instantiations working in parallel can process five times the data.

- **Agent-Chaining**—This is one of the most important techniques of which high-performance applications on the Tarari Processor can take advantage. In agent-chaining, the output from one algorithm (or Agent) can be used as the input for another algorithm, *while still staying on the board and without going back to the PCI bus.*

  Consider the example of pattern-matching in aerial photos, employing several different agents in the same Acceleration Agent Set. Compressed images are streamed onto the Tarari Processor and decompressed by the algorithm in the first agent. The resulting images are handed off to a pattern-matching algorithm in a second agent. For exact matches, a value is returned to the host application; fuzzy matches are handed off to a scoring algorithm in yet a third agent.

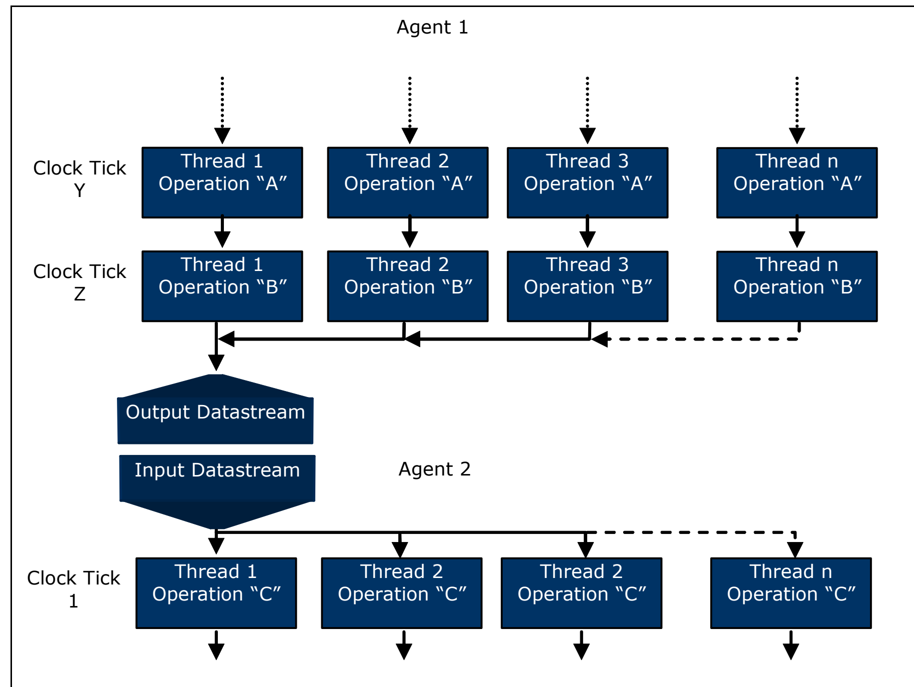


Figure 5--Agent-Chaining

Another example is that of a MIME-encoded, compressed e-mail attachment. In the Tarari Solution, one acceleration agent decodes the stream of data, then hands off its result to a waiting decompression agent. Once all processing is finished, the encoded, decompressed data are sent back to the host application.

For example, Acceleration Agent Sets for cryptography could perform tasks such as the RSA, SHA1, and 3DES algorithms, both pipelined (to execute all stages of all algorithms) and in parallel (through multiple instantiations). Dedicating these tasks to the Tarari Processor not only offloads them from the host processor, but also accelerates the processing of encrypted traffic on the host system.

Zero Bus Turnaround, Synchronous Static RAM (ZBT SSRAM)

Each Content Processing Engine has two Zero Bus Turnaround SSRAM modules available as low-latency, high-speed access to scratch-pad memory during operations. These SSRAMs are also available for caching local variables. Each of the two 18-bit SSRAMs can be accessed separately or combined for 36-bit operations.

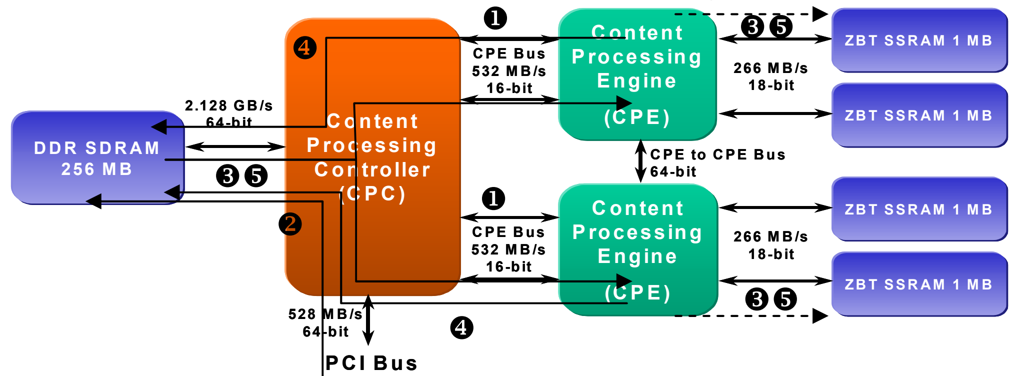
Double Data Rate, Synchronous Dynamic RAM (DDR Memory)

While the Content Processing Controller manages the movement and processing of data on the Processor, it uses Double Data Rate, Synchronous Dynamic RAM as its workspace. DDR memory is used for:

- high-speed Direct Memory Access (DMA) transfers between the host processor and the Tarari Processor over the PCI bus
- local storage for data being processed
- storage for acceleration Agents, to allow for immediate reconfiguration of logic

Workflow on the Tarari Processor

So, a typical application will flow as follows:



1) At system startup, the Agent Configuration Manager sends one or more Acceleration Agent Sets into DDR memory. The Content Processing Controller then loads, or instantiates, them into the Content Processing Engine(s). This is analogous to the Controller “flashing” the Engines with images of the agent sets.
2) The application on the host processor sends data over the PCI bus to the Processor, and the Controller stores the data temporarily in DDR memory.
3) The agent in the Engine fetches data across the Controller from DDR memory using one of three memory access modes (Direct, Streaming, and Transaction-based) and executes the algorithm on the data. The Engine may use Zero Bus Turnaround memory for lookup tables and as scratch-pad. If the agent takes advantage of parallelism, the Engines will execute different parts of the algorithm simultaneously.

4) The agent sends the result of the calculations back across the Controller to DDR memory.
5) If the agent takes advantage of pipelining or agent-chaining, another agent can now operate on the result by again calling it from DDR memory; otherwise, the Controller returns the result over the PCI bus to the host processor.

Evaluating Applications for the Tarari Processor

Applications that benefit the most from the Tarari Processor adhere to all or most of the following "A-B-C-D" guidelines, detailed below:

- A—Algorithm Acceleration—The algorithms run faster on the Tarari Processor than on the host processor.
- B—Benefits of Offloading—The overall system runs faster if some algorithms are offloaded.
- C—Compatibility of Hardware Platform and Algorithm—The algorithms are suitable for instantiation into the Tarari Processor.
- D—Dynamically Reconfigurable Hardware—The application or algorithms have functionality that changes over time.

A—Algorithm Acceleration

BENEFIT: By implementing one or more algorithms in hardware, the Tarari Processor can increase the overall performance of an application by accelerating those algorithms, and offloading them from the host processor.

METRIC: If the round-trip time between the host processor and the Tarari Processor is less than the time in which the host processor can process the algorithm by itself, the application is a candidate for the Tarari Solution. So, given that

$$\frac{\text{Algorithm processing time on host processor}}{(\text{Algorithm processing time on Tarari Processor + PCI bus transfer time})} = \text{Factor of acceleration}$$

the higher the Factor of acceleration, the better suited the application to the Tarari Solution. However, the overall acceleration of the entire application will only increase as the percentage of the accelerated algorithm increases when compared with the total algorithm or application.

For example, if we accelerate 20% of the total algorithm infinitely fast (so that it takes zero time to execute), then the entire algorithm will run in only 80% of the original time: a 25% improvement in performance. In the case of an accelerated algorithm being 50% of the total algorithm or application, the maximum possible increase would be 100%; i.e., the program runs twice as fast. Of course, the time to execute cannot reach zero, so actual performance improvements are also a function of the code efficiency on the reconfigurable logic.

Determining the Algorithm Processing Time

Estimating the algorithm processing time on both the host processor and the Tarari Processor requires careful engineering analysis, because it is dependent on many variables in the hardware and software design. Cycles-per-byte values for a variety of commonly used algorithms and applications (3DES, SHA-1, MD5, variants of AES and RC4 are used to protect online transactions) appear in Figure 6--Selected Algorithms—CPU Cycles/Byte Ratio, and clock cycles can also be measured while running algorithms obtained from standard libraries. The cycles-per-byte ratio represents the "compute-to-communicate" relationship at work, since all algorithms require some combination of both factors.

Calculating the PCI Bus Transfer Time

This example shows how to calculate the bus transfer time for a specific host platform. It makes these assumptions about the hardware characteristics of the platform being evaluated:

- 64-bit/66MHz PCI bus
- 50% PCI bus efficiency
- 2.4GHz CPU

The following calculations lead to a net cycles-per-byte ratio:

Step	Calculation	**Result**
PCI bus (64-bit/66MHz) throughput	64 bits/cycle x 66 million cycles/second	4,224 million bits/second
Equivalent in bytes	4,224 million bits/second ÷ 8 bits/byte	528 million bytes/second
Cycles required to transfer one byte (2.4GHz CPU)	2.4 billion cycles/second ÷ 528 million bytes/second	4.5 cycles/byte (approximate)

Assuming the output is approximately the same size as the input—see "Algorithm Input-to-Output Ratio" below—this figure must be doubled, because the transaction is not complete until the data travels in both directions: to the Tarari Processor, and then back to the host processor. This would result in a figure of 9 cycles per byte if the PCI bus ran at 100% efficiency.

However, given the overhead involved in preparing and sending data across the bus, including generating host interrupts, a PCI bus-efficiency of 50% is much more realistic. The result in this example is, therefore, 18 cycles to transfer one byte of data round-trip across the PCI bus; so for an algorithm to be a good

candidate for acceleration, that algorithm must execute in hardware at least 18 cycles per byte faster than the host processor could execute the algorithm (just to account for PCI bus transfers). If the data is streaming on and off the Tarari Processor, then the impact of this calculation is reduced as the latency for the calculation is a fixed amount of time.

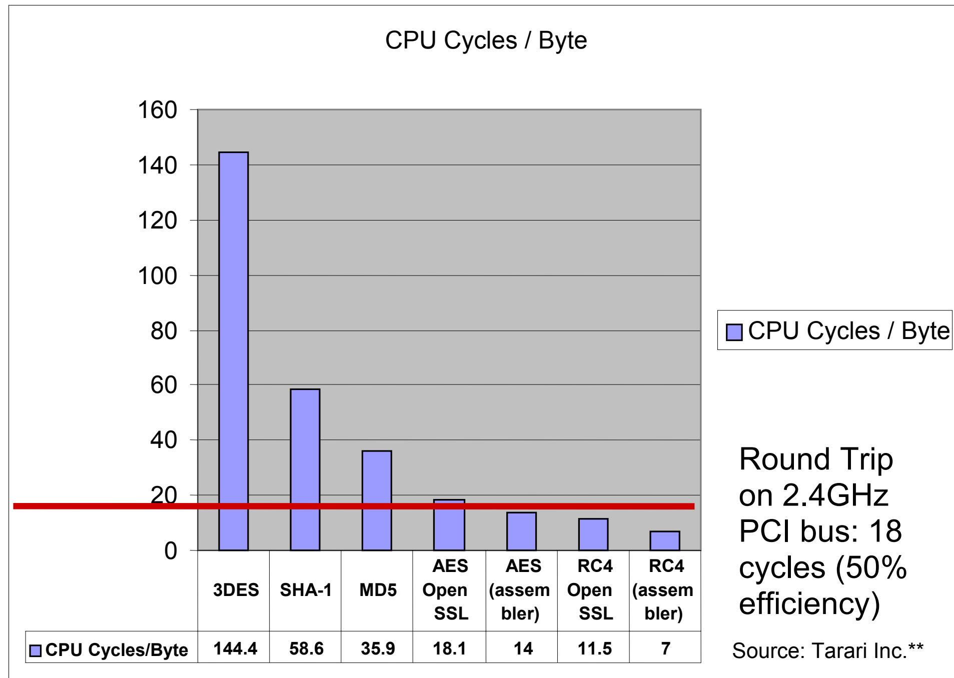


Figure 6--Selected Algorithms—CPU Cycles/Byte Ratio

Another paradigm is that of "compute vs. communicate". Figure 7--Algorithm Types—Compute vs. Communicate maps a range of commonly used algorithms by comparing compute cycles and bandwidth. The 3DES algorithm, for example, is a good candidate for acceleration, because the encryption processing typically runs on 90% of the data. The RSA algorithm is an even better candidate because its complex calculations require simultaneous multiplication of very large numbers. These calculations might consume 80% to 90% of the host processor's total compute cycles.

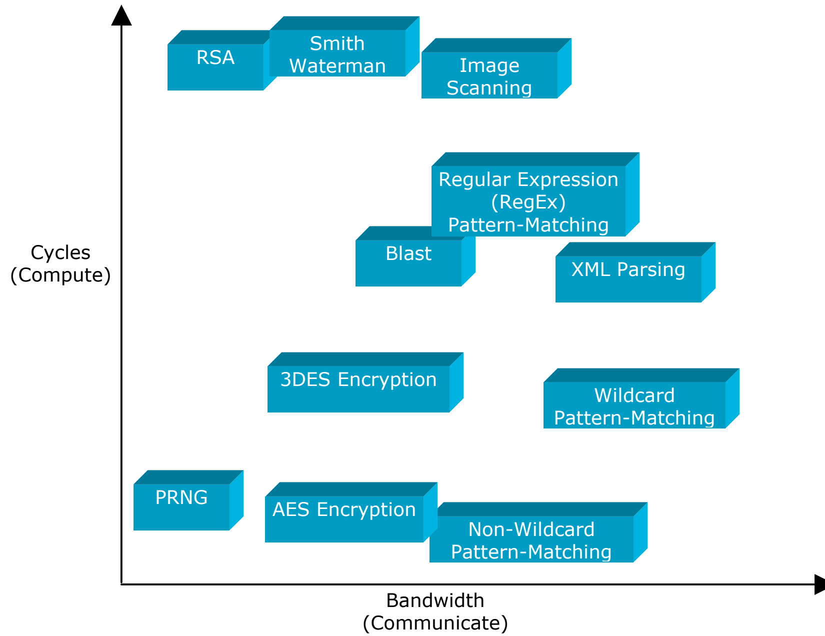


Figure 7--Algorithm Types—Compute vs. Communicate

Other Factors

Algorithm Input-to-Output Ratio

For some algorithms, all data is transferred to the Tarari Processor, but once the data is processed there, the Tarari Processor passes back only a short result. In virus scanning, for example, a large stream of data crosses the bus from host memory to DDR memory for processing, and a relatively small quantity of data (the result of the scan) is returned with only an indication of whether the data is free from viruses. For this type of application, a transfer rate of up to 2 Gbps is possible. For other algorithms, the entire data set must be transferred both to and from the Tarari Processor; thus, the practical maximum rate of transfer is just over 1 Gbps. So it is important to understand for each algorithm what quantity of data is returned for each byte of data that is transferred to the Tarari Processor.

Parallel Processing Techniques

Parallel processing techniques increase performance by instantiating multiple copies of an algorithm, or a portion of an algorithm. If, for example, a section of an algorithm operates much more slowly than the rest of the logic, then splitting the data path into two or more paths allows that section to be replaced with multiple copies of itself. Assuming that the Content

Processing Engines have sufficient resources to accommodate more than one copy, two copies of an algorithm in simultaneous execution will finish the operation in half the time. It is necessary to analyze the algorithm to determine whether it would benefit from running multiple processes in parallel.

Pipelining/Agent-Chaining Algorithms

An algorithm well suited to the Tarari Solution will buffer input/output, and balance input/output and computation by taking advantage of pipelining and agent-chaining operations on the Tarari Processor, as described in "Content Processing Engine" above.

Data Width

In some instances, wide data operations can be performed much faster on the Tarari Processor where there are no hard limits to the data width, as there are in the CPU. This allows the operation to be completed in one cycle on the Tarari Processor, instead of multiple cycles in the CPU. Alternatively, the Tarari Processor can also operate on smaller or odd data widths, such as 8 or even 13 bits.

Key Questions

- Does the application require a given level of data throughput, input bandwidth or output bandwidth? Is this a real-time application in which bandwidth is a significant component of the processing time? Can the PCI bus provide this bandwidth?
- Does the application have a high operation/byte ratio?
- Can the application benefit from parallel or multi-threading operations by running multiple instantiations on the Tarari processor?
- Can the application be pipelined? Is the next data operation dependent on the result of the previous data operation?
- From the time that the first data are moved from host to PCI bus, how long will it take to process the data and return the answer? Can input/output be overlapped with processing?
- Can the application take advantage of wide/odd data widths to offload more operations to the Tarari Processor?

B—Benefits of Offloading

BENEFIT: Host resources are freed up to perform other operations, such as feeding even more data to the Tarari Processor.

METRIC: Offloading is beneficial when the host processor can perform other work while the Tarari Processor executes the algorithm. The best candidates for offloading are those

algorithms that not only execute much faster in hardware, but also consume a large percentage of host processor cycles.

Freeing Up the Host Processor

It can make sense to offload algorithms *even if there is no acceleration*. Assume, for example, that an application uses 100% of the host processor's cycles, and a compression algorithm uses 50% of those cycles. If the compression algorithm were offloaded, then all of the host processing cycles would be available, roughly doubling the performance of the portion of the application that remained on the host processor. It is also possible to design the agent and its software driver to overlap input/output transfers with processing to overcome any PCI bus overhead.

Taking Advantage of No Data Dependencies

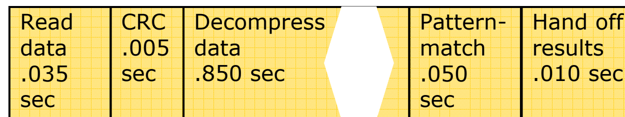
Overall system performance is also affected when the host processor algorithms have data dependencies on the offloaded algorithms. If there are no dependencies, then the algorithms can run simultaneously on host and Tarari Processors and there is an opportunity to increase overall performance of the system. Furthermore, if the application allows for overlapping I/O and computation, and does not need to pause for the result of a calculation being performed on the Tarari Processor before it can continue executing, then performance benefits due to freeing up host processor cycles can be even greater.

Calculating Overall System Acceleration

The total increase in system performance is a function of *both* the amount of acceleration delivered by the Tarari Processor *and* the percentage of CPU resources freed up as a result of offloading the work to the Tarari Processor. To determine the benefit that an application can realize from using the Tarari Processor to "redeploy cycles" in this manner, it is important to analyze the algorithm to see how much faster it runs in hardware than it does on the host processor. Consider, for example, a decompression operation which takes one full second to process on the CPU, and which requires 80% of the CPU's resources:

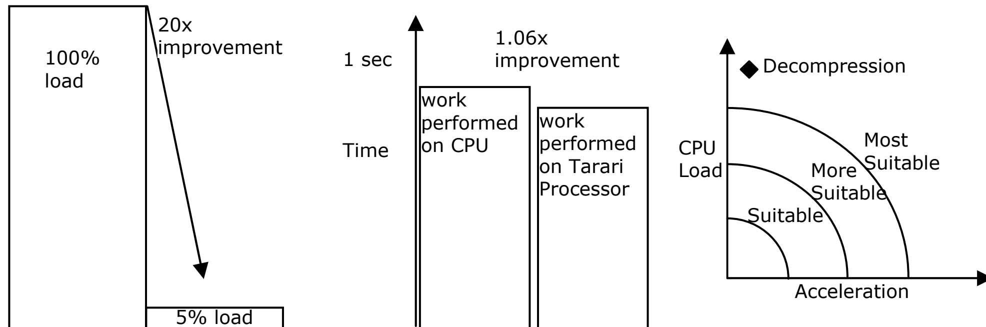
Read data .035 sec	CRC .005 sec	Decompress data .900 sec	Pattern-match .050 sec	Process results .010 sec

With the operation offloaded to the Tarari Processor, that same second of CPU time could look like this:



So, while the decompression work itself on the Tarari Processor went only slightly faster (.850 second instead of .900 second, or a 1.06x improvement), the CPU reclaimed .950 second (20x improvement), which it could devote to other operations. These savings combine for a 21x improvement in overall system performance.

As CPU load shifts to the Tarari Processor...	...and task completion time changes...	...suitability of application and overall system performance increase.



The following formula, based on Amdahl's Law[1], shows how to predict the overall system acceleration gained by accelerating a portion of the application, if the amount of acceleration and the fraction of host processor cycles that the algorithm consumes are known:

[1] According to Amdahl's Law, even as the performance of a single component of a system (or application) approaches infinity, system throughput will still exhibit the combined delays of the other components.

$$\text{Overall System Acceleration} = \frac{1}{(1 - \text{Fraction Accelerated}) + \text{Fraction Accelerated} / \text{Amount of Acceleration}}$$

In applying this formula to specific examples, it becomes clear how results can vary.

If 40% of an application can run 5 times faster, the system enjoys a 1.47x improvement:

- Fraction accelerated = .4
- Amount of acceleration = 5
- Overall system acceleration = 1/ [(1-.4) + (.4 / 5)] = 1.47

A second example, with a 5.26x improvement, shows that the best algorithms to accelerate are those which both consume a high number of processor cycles and run much faster in hardware.

If 90% of an application can run 10 times faster:

- Fraction accelerated = .9
- Amount of acceleration = 10
- Overall system acceleration = 1/ [(1-.9) + (.9 / 10)] = 5.26

Key Questions

- ❑ Does the application have an algorithmic kernel that represents a majority of the processing time? Can the algorithmic kernel be extracted from the application and offloaded to the Tarari Processor?
- ❑ Can the application take advantage of the freed-up cycles by executing additional tasks, such as post-processing the data from the previous kernel run, or preparing data for the next kernel run?
- ❑ Has the application been profiled to find out where the bottlenecks currently exist?

C—Compatibility of Hardware Platform and Algorithm

BENEFIT: The Tarari Processor is suitable for algorithm acceleration in most servers, appliances, and network devices with a PCI bus interface. The more a given compute-intensive algorithm can exploit the resources on a Tarari Processor, the greater the benefits to system performance.

METRIC: Evaluate the hardware and algorithm in light of these factors:

- System Bus
- On-Board Memory

System Bus

The Tarari Processor comes in a PCI form factor. The PCI implementation is a 32/64-bit, 33/66 MHz, half-length, 3.3v, PCI card. The host system must have one of these PCI slots available. Using a dedicated PCI bus provides the best performance results.

On-Board Memory

The Tarari Processor contains 256 megabytes of DDR memory, which provides a 64-bit/133MHz data path and is accessed through the Content Processing Controller. The DDR memory stores Content Processing Engine configurations and Content Processing Engine-specific data. The Content Processing Engine uses portions of the DDR memory for pre- and post-processed data for each algorithm. Tarari reserves 8 MB of this memory to store the Content Processing Engine configuration data.

Two Zero Bus Turnaround, Synchronous Static RAMs (ZBT SSRAM), organized as 512KB x 18, are also available to each of the Content Processing Engines. Each Content Processing Engine is connected to two Zero Bus Turnaround memory blocks by independent address and data lines operating at 133 MHz. The local Zero Bus Turnaround memory is typically used by algorithms that need a small amount of fast local memory, because it has a much lower latency than the DDR memory. The memory is available to the Content Processing Engine, and its use varies, depending upon the implementation of the algorithm. Developers have full access to the Zero Bus Turnaround memory as local temporary data storage for either the algorithm or scratch pad memory, where the requirements exceed what the block-RAMs alone can support.

Key Questions

- ❑ Does the server meet the Tarari Processor's physical requirements? Will the chassis take a full-height, half-length PCI card? Does the power supply provide 350 watts or more? Is there adequate ventilation for the 25+ watts of power dissipation? Does the fan have more than 20CFM throughput?
- ❑ There is 248MB of on-board DDR memory which can be split between the two Content Processing Engines. This memory will be used for input data storage, output data storage and possibly intermediate data storage. What is the input/intermediate/output data set size of the application? Does the data set fit within the 248MB of memory? If not, then can it be broken up into chunks that will fit? Can it be "streamed" in and out, in a systolic fashion, within the constraints of the DDR memory size?

- The Tarari Processor[2] features approximately 32 18Kbit block-(internal) RAMs that can be used as intermediate storage inside the Content Processing Engine. These block-RAMS have interfaces that allow low latency access to data, variables and coefficients. Does the algorithm/application make frequent use of any data that can take advantage of this feature?
- How much of the data processing is sequential in nature? How much of the algorithm requires random access to the data? If the algorithm requires random access, then will the data set fit into the on-chip block-RAM, or will it fit into the off-chip Zero Bus Turnaround memory? Is the data interface to the Zero Bus Turnaround memory wide enough to allow full processing speed?
- Is it possible to estimate the gate count (internal size requirements) based on the width of the operators and the number of arithmetic functions?
- Do most of the algorithm's operations require integer or floating-point math? If floating-point, then how much dynamic range and how many bits of precision are needed? Can the data be converted to integer? If so, then how many bits of precision are required?

D—Dynamically Reconfigurable Hardware

BENEFIT: Reconfigurable logic enables cost-effective updates in the event that protocols change and new standards emerge. This could extend the time-in-market of OEM products through multiple generations, by using the same core technology.

KEY TEST: If the hardware environment of the application is subject to change, or if changing traffic profiles prompt changes in the algorithms being accelerated, the Tarari Processor provides the market's most versatile solution.

An ASIC-based solution can quickly become obsolete with the first change to an industry standard. Whereas manufacture and update/upgrade of an ASIC solution will involve nonrecurring engineering (NRE) costs and substantial time to deploy—prototype, fault coverage, silicon design change, fabrication, verification, manufacture, shipment, installation—the Tarari Solution allows reprogramming of hardware using software tools and techniques, without requiring expensive field upgrade or production changes. Combining CPU technology with the flexibility of programmable logic enables OEMs to rapidly deploy solutions and to maintain those solutions more cost-effectively than their competitors can.

Furthermore, as the most burdensome of cycle-burning operations move from the host processor to the Tarari Processor, it soon becomes appropriate to re-examine the load on the host

[2] Versions 2.2 and 2.3.

processor for other compute-intensive operations. For example, if anti-virus and attachment scanning are the first applications offloaded to the Tarari Processor, anti-spam and intrusion detection will soon follow. With new agents pushed to reconfigurable logic in the Content Processing Engines, the Tarari Processor can accommodate such changes.

Key Questions

- Is it likely that the algorithm will evolve in depth or breadth? Is it subject to the requirements of users, customers, other applications or industry standards?
- After the first algorithm, will subsequent, compute-intensive operations be offloaded to the same Tarari Processor?
- Are multiple users likely to use the same equipment for different calculations? Can they effectively utilize the Tarari processor?

Algorithm Analysis Examples

1. Biotechnology Research

The computation requirements of genomic research are well known to be different from those of many other traditional HPC problems. The data is packed—2-bit or 5-bit representations of proteins or amino acids—and must be compared with the billions of other proteins or amino acids contained in the genomic database. This computation can be accelerated dramatically by passing the most compute-intensive processes to the Tarari Processor, as the following analysis (along the lines of the "A-B-C-D" outline of this paper) demonstrates.

A—Algorithm Acceleration

- The actual comparison comprises simple operations that are extremely compute-intensive and can be executed in hardware much faster than in software. A typical search with 5,000 query elements against a target database of 2 billion elements can take up to ten days to complete using a single 2GHz Xeon processor. This would have the Xeon processor performing 100 million comparisons per second.
- When offloaded to a 6 million-gate Tarari Processor, this application would typically reserve 2 million gates for "housekeeping," leaving 4 million gates available. Assuming that each comparison operation requires 20,000 gates, the Tarari Processor could perform 200 comparisons per clock cycle (4 million gates / 20,000 gates per comparison). Since the Tarari Processor runs at 100MHz, the processor could perform 200 comparisons/cycle x 100 million cycles per second for a total of 20 *billion* operations per second. This is roughly 200 times the speed of a single Xeon processor.

B—Benefits of Offloading

- There is other work that the host processor can perform while it awaits results from the Tarari Processor, notably, preparation of the query and target data for the actual search performed on the Tarari Processor and storage and representation of the finished data. For older systems with slower processors, this means that the Tarari Processor could be added to each node of a cluster with minimal incremental expense above the cost of the Tarari Processor.

C—Compatibility of Hardware Platform and Algorithm

- The operations involved can be easily and profitably instantiated in hardware on the Tarari Processor. In this example, every computational unit of the Tarari Processor

has been used and all of the block-RAM has been allocated to the algorithm. Because the data has been packed in bit-level representations, there are no bandwidth concerns for moving data between DDR memory and the Content Processing Engines, nor is there a bandwidth issue for the DDR-to-PCI bus.

D—Dynamically Reconfigurable Hardware

- The algorithm, depending upon its implementation, may take advantage of the ability to dynamically reconfigure the Content Processing Engines. The algorithm could be adapted to configure itself one way for large queries and another way for small ones, and could be dynamically changed based upon the size of the query.
- While certain biotechnology-related algorithms (e.g., Smith-Waterman) change relatively little over time, other algorithms and applications can exploit dynamically reconfigurable logic.

2. Cryptography Acceleration Agent Set

The overall performance of the Secure Sockets Layer (SSL) protocol can be accelerated dramatically by passing the most compute-intensive processes to the Tarari Processor, as the following analysis demonstrates.

Certain of SSL's characteristics are eminently suited to acceleration by the Tarari Processor, as gauged by the "A-B-C-D" outline of this paper:

A—Algorithm Acceleration

- SSL comprises some simple operations that are compute-intensive and can be executed in hardware much faster than in software.

B—Benefits of Offloading

- The operations to be offloaded consume a high percentage of processor cycles on the host processor.
- There is other work that the host processor can perform while it awaits results from the Tarari Processor.

C—Compatibility of Hardware Platform and Algorithm

- The operations involved can be easily and profitably instantiated in hardware on the Tarari Processor.

D—Dynamically Reconfigurable Hardware

- SSL is based on standards and algorithms that change over time.

RSA

RSA (Rivest-Shamir-Adleman) is a public-key cryptographic algorithm, which means data is encrypted using a public key, and decrypted using a different, secret key. RSA is widely used in modern cryptographic protocols, especially for the purpose of securely exchanging keys for faster bulk-encryption algorithms, which is its function in SSL.

The core operation of RSA is a simple but intensive arithmetical computation: modular exponentiation of moderately-large integer values. That is, for positive integers X, E and M, RSA requires computing the formula:

$$\mathbf{X^E \bmod M}$$

where, typically, **X** ≥ 1,024 bits.

So, to analyze the RSA component of SSL in light of this A-B-C-D model:

A—Algorithm Acceleration

Due to the complexity of performing modular exponentiation ("modexp") with several base-16 multipliers, it takes 2 million CPU cycles to perform a single 1024-bit RSA operation. That means that a CPU operating at 2GHz could complete 1,000 operations per second, assuming that the CPU was doing no other processing at the time. Also, RSA calls in 512 bytes and returns 130 for a total of 642 bytes. Assuming conservatively that only one-third of the cycles are devoted exclusively to the modexp operations—so, 666,000 cycles to process 642 bytes—the number of CPU cycles per byte (>1,000 cycles per byte) is much higher than the baseline for sending traffic across the PCI bus (18 cycles per byte). Therefore, RSA is a good candidate for acceleration.

The computation work in RSA takes place on the two Content Processing Engines. Because the RSA agent can take advantage of parallelism, each Content Processing Engine can accommodate up to four modexp engines set up by the algorithm. Each modexp engine has five processing elements, and each processing element performs two 16x16 multiplications and four 16-bit additions per cycle, so the result is 80 multiplications and 160 additions per cycle.

B—Benefits of Offloading

In spite of the compute-intensive modular exponentiation, relatively little data travels back and forth in RSA decryption. The algorithm brings 512 bytes in for decryption in two transfers of 256 bytes each, performs its computations (during which there is approximately 1.2ms latency), then returns two blocks of 65 bytes each. Server CPUs, already juggling priority among dozens

or hundreds of other processes, can save 70-80% of their cycles by offloading RSA decryption.

System performance, then, will benefit from offloading RSA to the Tarari Processor because of RSA's high demand for compute cycles and low demand for bandwidth.

C—Compatibility of Hardware Platform and Algorithm

There are forty 18x18-bit multipliers in each Content Processing Engine, of which capacity only 16x16 is needed for RSA. If no other agents need to use the multipliers, the overall performance of SSL is maximized by spreading the RSA operation across both Engines, allowing RSA to use up to eighty multipliers, as we have noted. One independent agent can be instantiated on each Engine and the host processor can individually control and load-balance the two agents.

Each of the two RSA agents uses under 1MB of DDR memory for input and output queues. The host processor supplies the base address of this region using an input/output write. The agent requires, in each Content Processing Engine:

Resource on Content Processing Engine	Number Required by RSA Agent	Percent of Total, Each Engine
18x18-bit multipliers	40	100%
18 kbit block-RAMs	8	20%
Lookup tables	7600	74%
Flip-flops	6600	64%
Digital clock managers and clock nets	2	25%

D—Dynamically Reconfigurable Hardware

While RSA is not subject to frequent change as a standard, there is a possibility of change in factors on which it depends. For example, the Tarari Solution can easily accommodate key lengths of up to 2048 bits for encryption and 4096 bits for decryption, adequate by today's standards. It could also accommodate an application that called for much greater key lengths, although this would require changes to the agent. By using software techniques and tools, a programmer could effect the hardware changes required, an option not open to the user of an ASIC-based solution.

More likely, a developer might want to load-balance different operations between the Content Processing Engines by changing the Acceleration Agent Set, or change the amount of resources on the Tarari Processor dedicated to RSA in order to add other agents. The built-in flexibility of the Tarari Processor also opens up to the developer the opportunity to

change the mix of running agents (bulk encryption, DES, anti-virus, XML, etc.) on the Content Processing Engines as traffic profiles change.

Appendix A—Summary of Guidelines

A—Algorithm Acceleration

- Accelerated parts of applications can be sub-divided and expanded to operate in parallel.
- Multiple algorithms can be pipelined/chained to make the best use of available resources.
- Bandwidth requirements of algorithm data and control plane functions do not exceed PCI bus bandwidth of 4.267 Gigabits per second.
- The application's optimal throughput cannot be achieved by a software implementation, but is possible to achieve the expected throughput with hardware acceleration.
- The algorithm requires a high number of host processor cycles to process each byte of data.
- There are no other bottlenecks that could prevent the application from running faster, even if some algorithms are offloaded to hardware.
- Data can be pipelined (the output of one algorithm can immediately be used as input to the next algorithm), or fewer bytes of data can be sent back to the host processor than are sent to the Tarari Processor.
- The size of a typical block of data is large, or multiple small blocks can be aggregated.
- The application can benefit from running multiple simultaneous threads.

B—Benefits of Offloading

- The application can tolerate the latency introduced by transferring data across the PCI bus.
- The host processor can perform other applications while waiting for the Tarari Processor to process offloaded data.
- The algorithm to be offloaded consumes a high percentage of host processor cycles relative to other tasks that the system is performing.

C—Compatibility of Hardware Platform and Algorithm

- The system has at least one 3.3v-signaling-level PCI slot, preferably a dedicated 64-bit/66 MHz slot.
- The algorithm can be implemented in the two Content Processing Engines of the Tarari Processor, using their internal and external resources.
- The algorithm to be offloaded can benefit from as much as 256 MB SDRAM + 4 MB ZBT SSRAM local data storage.

D—Dynamically Reconfigurable Hardware

- The algorithm has characteristics that are likely to change over time.
- The design will benefit from different algorithms being swapped in and out of the Content Processing Engines as data characteristics change.

Appendix B—Advanced Topics

Number of Gates and Configurable Logic Blocks Required

Each Content Processing Engine on the current Tarari Processor includes these hardware components[3]:

- 40 18x18-bit multipliers
- 40 18 kbit block-RAMs
- 1280 configurable logic blocks
- 4 clock managers

To determine whether an application or algorithm is suitable for acceleration by the Tarari Processor, estimate the hardware requirements of the algorithm. Estimating the extent to which a given algorithm can exploit these hardware requirements is a difficult task, but consider these factors:

1. Each internal block-RAM is configured in variable widths or 1, 2, 4, 9, 18, or 36 bits that can be concatenated for greater width and/or depth. Estimate the amount of "scratch pad" storage required for small look-up tables, FIFOs, stacks, rate buffers, register arrays, staging buffers, and complex data structures.
2. Estimate the required number of flip-flops by counting the storage registers needed to store intermediate and final results.
3. Find the number and width (number of bits) of major logic functions that the algorithm requires.
4. To account for management interface overhead, multiply each of the foregoing results by 1.3 to arrive at the total number of flip-flops and look-up tables required.

Maximizing the use of these resources, and exploiting the parallel processing and pipelining/agent-chaining features of the Tarari Processor, provides the best increases in overall application performance.

[3] The current release of the Tarari Processor uses Xilinx* XC2v1000 Field Programmable Gate Arrays. Future versions may have different characteristics.

Legal Information

Tarari is a trademark or registered trademark of Tarari, Inc. or its subsidiaries in the United States and other countries.

Information in this document is provided in connection with Tarari products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Tarari's Terms and Conditions of Sale for such products, Tarari assumes no liability whatsoever, and Tarari disclaims any express or implied warranty, relating to sale and/or use of Tarari products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright, or other intellectual property right. Tarari products are not intended for use in medical, life-saving, or life sustaining applications. Tarari may make changes to specifications and product descriptions at any time, without notice.

Copyright © 2002-2003 Tarari, Inc. All rights reserved.

* Other names and brands may be claimed as the property of others.

** Performance tests and ratings are measured using specific computer systems and/or components, and reflect the approximate performance of Tarari products as measured by those tests. Any difference in system hardware or software design or configuration can affect actual performance. Buyers should consult other sources of information to evaluate the performance of components they are considering purchasing. For more information on performance tests, and on the performance of Tarari products, contact us as indicated below.

High-Performance Computing Algorithm Analysis

A Tarari White Paper

Additional information: info@tarari.com
Internet: http://www.tarari.com/
Telephone: (858) 385-5131
Fax: (858) 385-5129

Tarari, Inc.
10908 Technology Place
San Diego, CA 92127-1874
USA