Spatio-Temporal Signal Twice-Whitening Algorithms on the hx3100™ Ultra-Low Power Multicore Processor

T. S. Humble, P. Mitra, and J. Barhen
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, Tennessee, United States of America
humblets@ornl.gov

J. Polcari
Science Applications International Corporation
Washington DC, United States of America

B. Schleck
Coherent Logix, Incorporated
Austin, Texas, United States of America

M. Traweek
Maritime Sensing Branch
Office of Naval Research
Arlington, Virginia, United States of America

***Abstract*—While modern signal detection theory fully accounts for spatially distributed sensors, exploiting these techniques for real-time sensing using large, underwater acoustic arrays requires advances in the spatio-temporal signal processing algorithms. In particular, the computational complexity of many spatio-temporal processing techniques is so large that conventional computer processors lack sufficient throughput to provide real-time processing of large spatio-temporal data sets. These limits are exacerbated when constraints, such as power consumption or footprint, reduce the available computational resources. In this report, we demonstrate an implementation of a signal twice-whitening algorithm that is better suited for processing spatio-temporal data in real time. We emphasize these advances by implementing data whitening on the Coherent Logix hx3100 processor, a programmable multicore processor intended for low-power and high-throughput signal processing. These results serve as an example of how the novel capabilities available from emerging multicore processor platforms can provide real-time, software-defined processing of large data sets acquired by spatially distributed sensing.**

sensor array processing; data whitening; multicore processors; spatio-temporal detection theory

I. INTRODUCTION

Underwater sensor arrays offer opportunities for the spatio-temporal discrimination of acoustic signals, e.g., when coupled with beamforming or adaptive processing techniques. The latter capability proves useful for the detection and tracking of underwater targets in noisy environments, where the phase-coherent, weighted addition of independent signals enables greater sensitivity to source locations. The number of sensing elements, or sensor channels, and their corresponding geometry are important factors in beamformer design, and the ultimate spatio-temporal solution will necessarily vary with geometry. Indeed, for mobile or configurable arrays, the ability to adapt the beamforming strategy according to the array geometry represents a fundamental step, e.g., toward optimal detection of a moving target [1].

In general, spatio-temporal processing techniques are computationally costly because of the large size of the collected data as well as the complexity of the underlying algorithms. The computational challenges only increase when constraints on the time-to-solution or the available computational power are imposed. A typical approach to enable fast and power efficient computation employs dedicated digital signal processing hardware, e.g., ASICs and FPGAs, hardcoded for a given signal processing strategy. These approaches, however, can necessitate long and costly development time and, therefore, there is an interest in alternative, software-defined techniques that would have faster development times.

One significant challenge to the development of software-defined processing has been that the throughput and energy efficiency of general-purpose processors do not compare well with ASIC or FPGA solutions. Recently, however, a shift in processor design to various multicore architectures have enabled a variety of emerging multicore processors to approach the raw performance and power consumption goals required for high throughput, power-efficient signal processing. Consequently, an outstanding question is whether conventional spatio-temporal processing algorithms can be implemented efficiency on these emerging processor platforms and subsequently utilized as for the real-time processing.

In this paper, we report the implementation of a spatio-temporal twice-whitening algorithm on an ultra-low-power, programmable multicore processor [1], [2]. Twice whitening is a means of decorrelating white noise components in an observed data set. By whitening the observed data with respect to the observed noise, the relevant signal can be decorrelated from the background environment, e.g., prior to signal detection. Whitening arises frequently within the context of independent component analysis for blind signal separation as well as in matched-filter detection strategies. Spatio-temporal twice-whitening refers to decorrelating the temporal noise in signals obtained, e.g., from large, spatially disperse, array sensors. As it is computationally demanding to perform twice whitening of large, arbitrary data sets on

978-1-4244-5222-4/10/$26.00 ©2010 IEEE

general-purpose processors, we demonstrate a reformulation of spatio-temporal twice whitening that reduces the computational complexity many orders of magnitude by exploiting the spatial structuring of the array data.

More important, we demonstrate that twice-whitening can be implemented on a Coherent Logix ultra-low-power processor [3]. Notably, the hx3100 processor is capable of a peak power efficiency of 16 GFLOPS/W and a peak performance of 25 GFLOPS. The results of this work support the prospect that future multicore processors will be capable of implementing fully software-defined acoustic signal processing techniques in the backend of a sensing array.

In Sec. II, the novel formulation of spatio-temporal twice whitening is presented, while Sec. III introduces the hx3100 processor and describes its key features. The implementation of twice-whitening on the hx3100 processor is presented in Sec. IV and a summary of the observed performance is provided in Sec. V. In Sec. VI, we discuss the implications of these results and offer final conclusions.

II. Spatio-Temporal Twice Whitening

Consider a collection of N_e acoustic channels having some associated spatial distribution. In addition, assume each sensor channel collects N_t complex-valued samples with the output defined in the temporal domain. This yields a total of $N_\tau = N_e \times N_t$ samples for the array which is represented collectively by the vector $\mathbf{r}$. The vector $\mathbf{r}$ is constructed from N_e ordered partitions, e.g., $\mathbf{r}^{(i)}$ represents data from the i^{th} channel, such that

$$\mathbf{r} = \begin{pmatrix} \mathbf{r}^{(1)} \\ \vdots \\ \mathbf{r}^{(N_e)} \end{pmatrix}. \tag{1}$$

The i^{th} partition of samples $\mathbf{r}^{(i)}$ corresponds to either the sum of a sought after signal and the associated noise, i.e.,

$$\mathbf{r}^{(i)} = \mathbf{s}^{(i)} + \boldsymbol{\eta}^{(i)}, \tag{2}$$

or to the absence of any signal and, therefore, only the background noise. A fundamental goal of detection theory is to discriminate between these two hypotheses [1].

Following Eq. (2), the spatio-temporal noise covariance matrix for the array of N_e sensor channels is represented by

$$\mathrm{H} = \begin{pmatrix} \mathrm{H}^{(1,1)} & \cdots & \mathrm{H}^{(1,N_e)} \\ \vdots & \cdots & \vdots \\ \mathrm{H}^{(N_e,1)} & \cdots & \mathrm{H}^{(N_e,N_e)} \end{pmatrix} \tag{3}$$

where

$$\mathbf{H}^{(i,j)} = \boldsymbol{\eta}^{(i)}\, \boldsymbol{\eta}^{(j)\dagger} \tag{4}$$

represents an $N_t \times N_t$ submatrix formed by the outer product of the noise vectors for the i^{th} and j^{th} sensor channels. Whitening of the signal vector $\mathbf{r}$ refers, mathematically, to the application of the square-root inverse of $\mathbf{H}$, i.e., $\mathbf{z}=\mathbf{H}^{-1/2}\mathbf{r}$. Twice-whitening, i.e.,

$$\mathbf{x} = \mathbf{H}^{-1}\mathbf{r} \tag{5}$$

forgoes the square root operation; this can prove sufficient, for example, when computing statistical descriptions of the signal where the whitening operation reduces mathematically to effect of twice-whitening, e.g., $\mathbf{z}^\dagger\mathbf{z} = \mathbf{r}^\dagger\mathbf{H}^{-1}\mathbf{r}$.

As an example of the dimensions attributed to these tensors, we note that for very large arrays the number of acoustic channel may be on the order $N_e \sim 10^3$. Assuming each sensor collects $N_t \sim 10^4$ samples, the resulting $N_\tau \times N_\tau$ noise-covariance matrix $\mathbf{H}$ has on the order of 10^{14} entries. As noted above, constructing the inverse of $\mathbf{H}$ is essential in the theory of the optimum receiver [1]. The computational complexity of inversion scales as $O(N_\tau^3)$, and, due to the symmetric positive semi-definiteness of $\mathbf{H}$, can often be efficiently obtained by Cholesky decomposition [4]. However, even then, direct calculation of the twice-whitened signal vector $\mathbf{x}$ is exceedingly expensive for very large arrays.

To overcome the poor scaling associated with direct computation of the inverse of $\mathbf{H}$, we reformulate the data-whitening algorithm to exploit the spatio-temporal organization of array data. First, we limit consideration of the noisy background to spatially diffuse noise that is wide-sense stationary. This imposes the restriction that the submatrices of $\mathbf{H}$ be Toeplitz and that the full matrix be block Toeplitz. Consequently, due to the Hermitian form of $\mathbf{H}$ following Eq. (4), the noise covariance matrix exhibits the block-diagonal structure illustrated in Fig. 1.

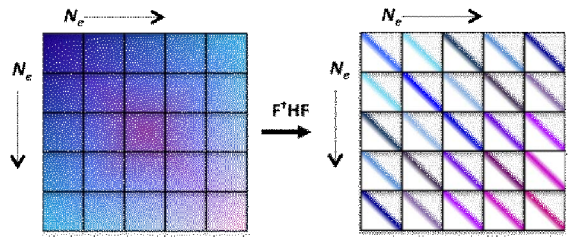

Figure 1. Block structure of the inter-channel covariance matrix $\mathbf{H}$. Symmetry arises from the condition of spatially diffuse noise sources. Note that the diagram neglects differences due to conjugations, e.g., $\mathbf{H}^{(i,j)} = \mathbf{H}^{(i,j)\dagger}$. The Fourier transform reveals the block-digaonal submatrix Λ.

The spectral theorem yields the eigenvalues of each $N_t \times N_t$ submatrix $\mathbf{H}^{(i,j)}$ with $i, j = 1$ to N_e, and the Fourier analysis of the full covariance matrix $\mathbf{H}$ may be factored as

$$\mathbf{H} = \mathbf{F}\,\boldsymbol{\Lambda}\,\mathbf{F}^\dagger, \tag{6}$$

where $\boldsymbol{\Lambda}$ is an $N_\tau \times N_\tau$ band-diagonal matrix composed of diagonal submatrices $\Lambda^{(i,j)}$ representing the eigenvalues of the corresponding noise covariance between channels i and j, and $\mathbf{F}$ symbolizes the block-diagonal form of the discrete

Fourier transforms applied to each block. The resulting structure of $\mathbf{\Lambda}$ is illustrated in Fig. 1. Elements of the matrix $\mathbf{\Lambda}$ then may be permuted to form a block-diagonal matrix

$$\mathbf{K} = \mathbf{\Omega}\,\mathbf{\Lambda}\,\mathbf{\Omega}^{T}, \tag{7}$$

where $\mathbf{\Omega}$ symbolizes the orthogonal permutation matrix and the k^{th} block $\mathrm{K}^{(kk)}$ is $N_e \times N_e$ for $k = 1$ to N_t. See Fig. 2.

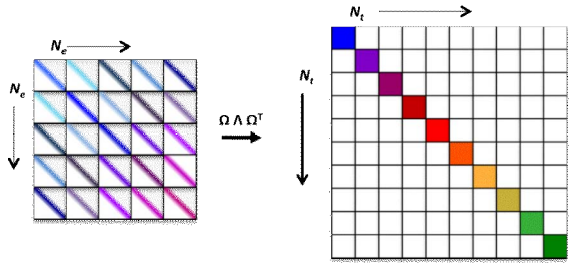

Figure 2. Permutation of the band-diagonal matrix Λ yields the block-diagonal matrix K, where each diagonal block $\mathrm{K}^{(kk)}$ is $N_e \times N_e$ in size.

The inversion of $\mathbf{H}$ can now be recast as

$$\mathbf{H}^{-1} = \mathbf{F}\,\mathbf{\Omega}\,\mathbf{K}^{-1}\,\mathbf{\Omega}^{T}\,\mathbf{F}^{\dagger}. \tag{8}$$

The cost of computing the inverse of $\mathbf{H}$ from the inverse of K is then reduced from $O(N_\tau^3)$ to $O(N_t\,N_e^3)$, where the inversion of the N_t submatrices of K each carry $O(N_e^3)$ cost. With respect to the resource estimates cited at the beginning of Sec. II, exploiting the spatial structure of the noise covariance matrix reduces the overall complexity of inversion by ~8 orders of magnitude.

Given the block-diagonal structure of K illustrated in Fig. 2, twice-whitening of the spatio-temporal data reduces to a system of N_t uncoupled equations, i.e., for $k = 1$ to N_t

$$\mathbf{K}^{(kk)}\,\mathbf{y}^{(k)} = \mathbf{d}^{(k)}, \tag{9}$$

where $\mathbf{y}^{(k)}$ and $\mathbf{d}^{(k)}$ represent $N_e \times 1$ ordered partitions of the transformed vectors

$$\mathbf{y} = \mathbf{\Omega}^{T}\,\mathbf{F}^{\dagger}\,\mathbf{x} \tag{10}$$

and

$$\mathbf{d} = \mathbf{\Omega}^{T}\,\mathbf{F}^{\dagger}\,\mathbf{r}, \tag{11}$$

respectively. Solutions to the latter equations are obtained to recover the twice-whitened signal vector $\mathbf{x}$. Denoting the Cholesky decomposition of $\mathrm{K}^{(kk)}$ as

$$\mathbf{K}^{(kk)} = \mathbf{L}^{(k)}\,\mathbf{L}^{(k)\dagger}, \tag{12}$$

with $\mathbf{L}^{(k)}$ a lower triangular matrix, then the twice-whitened partition $\mathbf{y}^{(k)}$ can be recovered from Eq. (9) by first solving for the intermediate result $\mathbf{b}^{(k)}$, defined by

$$\mathbf{L}^{(k)}\,\mathbf{b}^{(k)} = \mathbf{d}^{(k)}, \tag{13}$$

using forward elimination, before solving the subsequent equation

$$\mathbf{L}^{(k)\dagger}\,\mathbf{y}^{(k)} = \mathbf{b}^{(k)}. \tag{14}$$

This result can then be permutated and inverse transformed according to Eq. (10) in order to obtain the twice-whitened signal vector $\mathbf{x}$.

III. HyperX hx3100

A. *HyperX Architectural Overview*

The hx3100 processor is the latest entry in the HyperX family of ultra low power, massively parallel processors produced by Coherent Logix, Inc. The hx3100 processor is a 10-by-10 array of processing elements (PEs) embedded on an 11-by-11 array of data management and routing units (DMRs). At the typical system clock frequency of 500 MHz, the maximum chip-level throughput of for 16-bit integer operations is 50 GMACS. For 32-bit floating-point operations, 25 GFLOPS may be achieved. Alternatively, when power consumption is prioritized, performance can be measured as 32 16-bit GMACS/Watt or 16 32-bit GFLOPS/Watt. This translates into energy consumption on the order of 10 picoJoules (pJ) per instruction, and rivals that of dedicated ASIC designs.

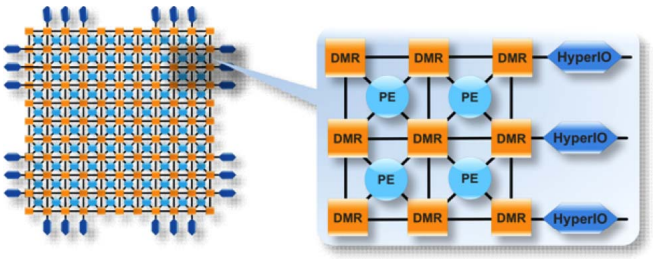

Figure 3. Layout of the Coherent Logix hx3100 processor, a 10-by-10 array of processing elements (PEs) inleaved by an 11-by-11 array of data management and routing units (DMRs). An exapnded view of four PEs surrounded by 9 DMRs demonstrates the degree of connectivity. HyperIO references the input/output ports used by the hx3100 processor to community with off-chip memory (DRAM) or other hx3100 processors.

The DMR network provides Direct Memory Access (DMA) for the PEs to both on-chip and off-chip memory. Each DMR has 8 kB of SRAM and operates at a read/write cycle rate of 500 MHz, while the eight independent DMA engines within a DMR may act in parallel. A PE directly accesses data memory in four neighbouring DMRs, such that 32 kB of data is addressable by any given PE. In addition, when a subset of PEs shares direct access to the same DMR, the associated memory space may act as shared memory between PEs. This inter-connectedness between cores and memory provides for a mixture of shared and distributed memory hierarchies. Data movement is managed by the programmer through a series of receive and send calls.

Each DMR consists of 8 16-bit DMA engines that route memory requests from neighbouring PEs and support routing memory requests managed by other DMRs. In addition to supporting on-chip DMAs, the DMRs handle requests to off-chip memory, including the eight DDR2 DRAM ports. Other off-chip IO ports are also available, including CMOS or LVDS interfaces. Moreover, the 24 IO ports surrounding (six per side) the chip can be wired to connect together multiple HyperX chips. In the latter context, the size of the computational resource would seamlessly scale with inter-chip routing handled by the associated IO controllers.

B. *Integrated Sofware Development Environment*

Programming the HyperX entails writing an ANSI C program, which defines the parallelism in the algorithm through the use of the industry standard Message Passing Interface (MPI) protocol.

The Coherent Logix software tools automatically assign individual tasks to PEs, and create routing to support the movement of data between PEs. Tasks may be both parallelized and pipelined across multiple cores, while DMAs are controlled explicitly in software. The latter steps provide opportunities for designing the flow of program execution such that resource constraints and programming requirements are met.

C. *Power Management*

The current version of the HyperX architecture provides the capability to power down quadrants of the PE grid that are unneeded by a designed application. As a result, programs can be optimized with respect to energy usage (of the chip) as well as the computational speed and memory bandwidth usage. Wake-up signals can be triggered by external events, such that the processor may be shutdown during period of computational idle time. These features, i.e., the extensible multi-core architecture and economical power consumption, suggest that the HyperX should be an interesting candidate for low-powered signal processing applications, such as long-range unmanned undersea vehicles.

IV. TWICE-WHITENING ON HX3100

The large number of individually programmable cores and multiple IO ports in hx3100 benefit our implementation of the twice-whitening algorithm discussed in Sec. II. Labels for the hardware components are defined in Fig. 3.

A. *Program Design*

Our implementation of twice-whitening on the hx3100 organizes the algorithm into multiple programs occupying multiple computational cores. Refer to Fig. 5 for the location of the program described below. First, prior to any data processing, simulated input data is loaded into off-chip DRAM memory banks. Here, the data represents single-precision, complex-valued signal vectors stored in sample-consecutive order in DRAM 1. At runtime, the program IOPE 1 fetches one input vector from DRAM 1 and copies the data onto the chip via a series of 4 routing programs labeled RPEs. The latter programs manage and synchronize the input data for a complex-to-complex FFT program.

Upon completion of the FFT of one input vector, the Router programs request IOPE 2 to store the transformed data off-chip in DRAM 2. Storage of the intermediate results is required to free on-chip memory for processing the FFT of the next input vector. The copying process occurs directly from the DMRs used by the FFT cores and, specific to our algorithm, requires a substantial amount of processing time. This overhead results from the use of a write operation in which each complex-valued sample is written to DRAM with a stride of N_e. The purpose of this strided write operation is to perform the permutation denoted by Ω in Sec. II. A diagram illustrating this point is shown in Fig. 4.

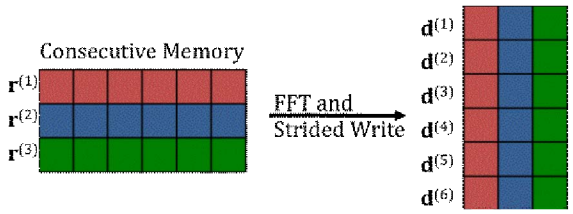

Figure 4.Illustration of the how input vectors are sequentially transformed and then permutated using a sample-by-sample strided write to DRAM. As a result, the vector $\mathbf{d}^{(k)}$ read directly from DRAM can be immediately used for the twice-whitening operation defined by Eq. (9).

Although this method of permutating the data requires a nontrivial amount of the processing time, the cost per input vector is independent of the stride size N_e. By comparison, approaches that permutate the input vector on-chip should be expected to have processing times that scale, at best, linearly with N_e. Ultimately, such an approach would reach a natural limit due to the finite size of on-chip memory. Indeed, for the size of the simulated data set, cf. Sec. IIB, this approach is not possible.

Once all N_e input vectors have been transformed and written to DRAM 2, IOPE 2 changes state and begins to fetch the permutated, length-N_e vectors that represent the partitions $\mathbf{d}^{(k)}$ introduced in Eq. (11). Simultaneously, IOPE 3 fetches the corresponding, precomputed Cholesky decomposition from DRAM 3. The latter is represented by the N_e (N_e – 1) / 2 elements of the upper triangular, complex valued matrix obtained by an offline decomposition of the matrix $\mathbf{K}^{(kk)}$. Future versions of this program will implement the Cholesky decomposition of each $\mathrm{K}^{(kk)}$ matrix alongside the transformation and permutation of the signal vectors. Both sets of data are moved to the DMRs of the Whitening program. It is within this program that that forward elimination (13) and back substitution (14) are used to solve Eq. (9), both requiring $O(N_e^2)$ operations to complete. The resulting output of this program is then written back to DRAM 4 via IOPE 4. Again, a strided-write operation is used, with the N_e components of the output spaced by N_t positions, so that once all N_t solutions are obtained, DRAM 4 stores, in consecutive order, the N_e partitions of $\mathbf{y}$ defined in Eq. (10). Ultimately, upon completion of the program, this output is written to file for post-processing analysis.

An important programming consideration for this platform is the synchronization between individual cores to ensure continuity and correctness of data movement. The 8 kB memory available from each DMR necessitates precise accounting for the size, location, and timing of each data segment. This inter-core communication is performed using a proprietary API written for the C language. For example, blocking variants of DMA send and receive functions provide a basic communication mechanism for data synchronization, while similar techniques are available for reading and writing to DRAM.

The placement and routing of individual processing tasks, e.g., the input server or the FFT cores, with respect to the layout of available cores is also an important consideration for program performance. While this level of detail can be programmed explicitly, the accompanying development environment, cf. Sec. III.B, provides a global optimizer and scheduler to perform placement and routing. For example, routing constraints ensure the IO server programs are placed adjacent to the off-chip IO ports. Similarly, placement and routing ensures that on-chip DMA routes do not collide, e.g., transferring data between PEs.

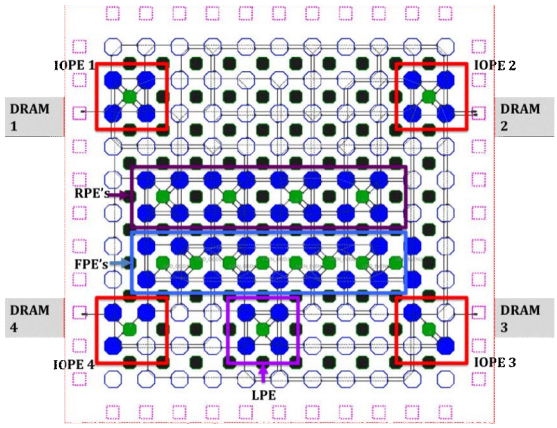

Figure 5. Layout of the twice-whitening program on the hx3100 processor. Boxes circumsscribe the participating PEs and relevant DMRs for each individual program.

B. *Program Results*

As implemented, the full program processes a series of 8192-complex-point input vectors, each representing a noisy sinusoidal typically of a monotone acoustic source collected against a diffusive (thermal) background. When running on both the simulator and the hx3100 processor, multiple input vectors are loaded into one of the off-chip DDR2 memory banks. Vectors are stored in deinterleaved, bit-reversed order, with a single input vector representing 64 kb of memory. Eight DMA transfers are required to load a collection of 8, on-chip DMR elements. This IO service requires 1 core for managing request to the DDR2 and, in a double buffering approach, 3 neighboring DMRs to optimize timing of the data transfer.

An 8192-point, complex-to-complex decimation-in-time FFT code is used to transform the input vector data. Each instance of the FFT employs 8 processing elements. The FFT algorithm requires bit-reversed-order input data, and this reordering of the input is done prior to loading the DRAM. The subsequently transformed output is transferred via DMA to an output server core that writes the data to DRAM 2 using a second off-chip IO port. The input and output servers are synchronized to avoid overwriting data stored in the FFT processing cores.

We have implemented the above program on the HyperX hx3100, and we have profiled the program elements using the Integrated Software Development Environment. The results below refer to version 3.0.1 of the HyperX ISDE tools for the case of eight 8192-pt complex samples.

Table 1. Summary of program elements and resources used, including the number of PEs, the number of DMRs, and the number of cycles per processed quantity.

Program	PEs	DMRs	Cycles
IOPE 1	1	4	374,94/vector
RPE's	4	16	N/A
FPE's	8	18	137,741/vector
IOPE 2 (write)	1	4	565,749/vector
IOPE 2 (read)	1	1	136/vector
IOPE 3	1	2	358/vector
LPE	1	4	4,384/vector
IOPE 4	1	3	539/vector
TOTAL	17	52	42,014,925

The cycle counts recorded for individual program elements are shown in Table 1. For the purposes of bookkeeping, we identify the number of resources contributing to each program. For example, the resource contributing to the FFT, i.e., the FPEs, consist of the 8 PEs and 16 DMRs, cf. Fig. 5. Similarly, 4 PEs and 16 DMRs comprise the Routing program (RPEs). The remaining program elements utilize a single PE and various numbers of DMRs.

The number of cycles normalized to the relevant input type are shown for each program in Table 1. For example, the first input/output server, IOPE 1, requires 37,494 clock cycles to read one input vector from DRAM 1 and synchronize transfer of that vector to the RPEs memory space. This accounting of clock cycles highlights the processing bandwidth associated with each program; data transfer is not computationally complex, but a moderate amount of time is needed to bring the data on to the chip. Regarding IOPE 1, note that the reported clock cycles includes sending the data to the RPE's, while the clock cycles reported for the FPEs incorporates the read operation from the RPE memory space. Thus, no clock cycles are reported for the RPE program as this is their sole function (apart from some minor transfer scheduling instructions).

For IOPE 3 serves to read the transformed and permutated vectors $\mathbf{d}^{(k)}$ from DRAM 3; recall the latter vectors are identified by the sample index k and, therefore, there are a total of 8192 vectors to process. The other program elements have similar interpretations of the reported cycle counts.

From Table 1, we note that IOPE 2 represents the most costly operations, per iteration, in the implementation. As

stated in Sec. IV.A and illustrated by Fig. 4, IOPE 2 is performing a sample-by-sample strided write of the FFT output. Many samples (8192) must be processed this way and the cost associated with strided writes to DRAM is high. More important, the IOPE 2 write operation represents a significant bottleneck in this implementation as the write operation takes significantly more time than the FFT processing. However, as noted earlier, the size of the processed data set (8 × 8192 samples = 512 KB) prohibits the possibility of performing the transposition step within the on-chip memory space. The cost of the strided write operation employed here is constant with respect to the number of input vectors.

Table 2. Program power consumption with respect to PE clock frequency and operating voltage.

	Tunable PE Clock Frequency (MHz)			
Voltage(V)	200	300	400	500
1.00 V	914.29	1162.15	1410.01	1657.89
0.95 V	865.95	1089.65	1313.35	N/A
0.90 V	820.09	1020.86	1221.64	N/A

Table 2 reports the total power consumption for the program with respect to the tunable PE clock frequency and operating voltage. The clock frequencies, tuned here from 200 MHz to 500 MHz, determine the overall time-to-solution of the program. Based on the total clock cycles reported in Table 1, this implies a range of 84.2 ms at 500 MHz to 210 ms at 200 MHz. Lowering the operating voltage, from 1.0 V to 0.95 V to 0.9 V, provides corresponding decreases in the power usage. A two-fold increase in power occurs between the case of 0.90 V at 200 MHz and the 1.0 V at 500 MHz. Depending on the specific application and performance requirements, twice-whitening on the HyperX can be tuned to meet satisfy both power and timing constraints.

V. CONCLUSIONS

We have presented an implementation of spatio-temporal twice-whitening on the hx3100 processor. We have been motivated by the prospects of performing real-time signal processing of spatio-temporal data. Our implementation of twice-whitening has exploited the properties of spatially diffuse noise to reduce the complexity of the twice-whitening algorithm form $O(N_t^3\ N_e^3)$ to $N_t \times O(N_e^3)$. This reduction in complexity can amount to many orders of magnitude improvement. We have also demonstrated a programmed implementation of this twice-whitening algorithm on the ultra-low-power hx3100 processor. The hx3100 processor provides a platform capable of simultaneously handling multiple instruction multiple data (MIMD) processing and an ability for block-stream processing of the simulated acoustic data. Moreover, the high throughput, low-power consumption features of this realization suggest an opportunity to employ adaptive signal processing techniques in reprogrammable platforms of relatively small footprints and power conscious designs.

ACKNOWLEDGMENT

This work was supported by the United States Office of Naval Research. Oak Ridge National Laboratory is managed for the US Department of Energy by UT-Battelle, LLC under contract DE-AC05-00OR22725.

REFERENCES

[1] H. L. van Trees, Detection, Estimation, and Modulation Theory, Vol. 1, New York: John Wiley & Sons, Inc., 2001.

[2] R. N. McDonough and A. D. Whalen, Detection of Signals in Noise, San Diego, CA: Academic Press, 1995.

[3] www.coherentlogix.com

[4] G. H. Golub and C. F. Van Loan, Matrix Computations, Baltimore, ND: John Hopkins University Press, 1996.