Science Journal of Circuits, Systems and Signal Processing
2013; 2(2) : 29-36
Published online April 2, 2013 (http://www.sciencepublishinggroup.com/j/cssp)
doi: 10.11648/j.cssp.20130202.12

Parallel implementation of the wideband DOA algorithm on single core, multicore, GPU and IBM cell BE processor

Mohammad Wadood Majid*, Todd E. Schmuland, Mohsin M. Jamali*

Department of Electrical Engineering & Computer Science, University of Toledo, Ohio, USA

Email address:
mohammad.wadoodmajid@rockets.utoledo.edu (M. W. Majid), mohsin.jamali@utoledo.edu (M. M. Jamali)

To cite this article:
Mohammad Wadood Majid, Todd E. Schmuland, Mohsin M. Jamali. Parallel Implementation of the Wideband DOA Algorithm on Single Core, Multicore, Gpu and Ibm Cell be Processor, *Science Journal of Circuits, Systems and Signal Processing.*Vol. 2, No. 2, 2013, pp.29-36. doi: 10.11648/j.cssp.20130202.12

Abstract: The Multiple Signal Classification (MUSIC) algorithm is a powerful technique for determining the Direction of Arrival (DOA) of signals impinging on an antenna array.The algorithm is serial based, mathematically intensive, and requires substantial computing power to realize in real-time.Recently, multi-core processors are becoming more prevalent and affordable.The challenge of adapting existing serial based algorithms to parallel based algorithms suitable for today's multi-core processors is daunting. DOA algorithm has been implemented on Multicore (Intel Nehalem Quad Core), NVIDIA's GPU GeForce GTX 260, and IBM Cell Broadband Engine Processor. This is in an effort to use DOA for real time applications. The DOA algorithm has been parallelized, partitioned, mapped, and scheduled on Multi-Core, GPU, and IBM Cell BE processor.The parallel algorithm is developed in C# for Intel Nehalem Quad Core, a combination of C and CUDA for GPU, and C++ for IBM Cell processor. The algorithm has also been implemented on single core for comparison purposes. Wideband DOA algorithm is implemented assuming 16 and 4 sensors using Uniform Linear Array (ULA).

Keywords: Direction Of Arrival (DOA), Single Core, Multicore, GPU And IBM Cell BE Processor

1. Introduction

Digital processing of radar signals at a very high speed is a necessity for military and civilian applications.Combining radar signals with innovations in image processing to extract more information from data or combine with other intelligent databases would be the norm for military applications.Often times these systems need to be small and mobile, but need to process more data.The latest computer system architectures being developed by the computer industry offers more processing power in a smaller footprint.

Parallel computing has been rejuvenated with the explosion of multicore technologies such as Multicore Processors, IBM Broadband Cell Processor, Many-Cores, Multicore DSPs, and General Purpose Graphic Processing Units (GPGPUs). Multicore technologies have also given a boost to parallel programming languages. Most multicore vendors are providing their own parallel programming language support for their processors. Parallel programming can be accomplished via Compiler based, Application Programming Interface (API), and Model-Of-Computation approaches. Therefore there is no universal way of implementing algorithms in parallel. Programmers are relying on mostly hand crafted approaches. It is also perceived that programming in parallel is difficult.

In order to fully exploit available parallel computation power, the algorithm is partitioned into an optimal number of parts and then to tasks.A task is a simple set of threads that can be run on a core. Tasks can be stored and are scheduled by a task scheduler onto a pool of kernel hardware threads. Partitioning of tasks is a challenging feat and requires optimization. There are opposing forces at work.too many partitions add overhead, and too few partitions leave processors idle, requiring optimal partitioning [1].

In most of the signal processing applications, main interest is to obtain an accurate estimation of the Direction-Of-Arrival (DOA) [2][3] of a received signal. The Multiple Signal Classification (MUSIC) algorithm [3] based on subspace techniques is one of the most popular algorithms due to its high resolution and the reduced computations compared to the Maximum Likelihood (ML) based algorithm.The most widespread wideband method is Coherent Signal-Subspace (CSS). When the incoming signal is in wideband then large amount of data is needed and more complex, computationally intensive wideband processing techniques will be required. The MUSIC algorithm for wi-

deband Directional of Arrival (DOA) has been proposed by Wang &Kaveh [3]. The MUSIC algorithm separates the wide frequency band into narrowband components [3].A focusing matrix is formed using this initial DOA estimate.The data set for this algorithm is divided into 64 segments and each segment contains 64 samples [3].A uniform linear array of sixteen elements is assumed.The wideband MUSIC algorithm has the following computational steps.

- Compute 64 sets of 64-point FFT
- Compute 33 covariance matrices (16 by 16)
- Compute eigenvalues and eigenvectors via Householder transformation and QR decomposition
- Compute DOA using Power Method
- Use this DOA as an initial estimate
- Compute focus matrix
- Use this focus matrix as new covariance matrix
- Compute eigenvalues and eigenvectors via Householder transformation and QR decomposition
- Compute the Akaike Information Criterion (AIC) [4]
- Compute DOA using Power Method

It can be seen that the above MUSIC algorithm requires computation of the DOA algorithm two times. The entire MUSIC algorithm will require basic computational blocks of Fast Fourier Transform (FFT), covariance matrix, Householder decomposition, QR transformation, Akaike Information Criterion (AIC) [4], and Power Method.Each part is analyzed to explore any kind of dependencies between tasks. Each part is then parallelized.

2. Parallel Hardware

2.1. Intel Nehalem Quad Core

Homogeneous multicore stands out among a confluence of the current hardware trends as they provide an effective solution to manage the power and computation speed. A computationally intensive problem needs to be divided into discrete parts for concurrent computation. The Intel Nehalem Quad Core processor [5] is one of the parallel processing platforms that can be exploited for fast computation of algorithms for real time applications. Intel Nehalem Quad Core has four physical cores. Each physical core has two virtual processors running in parallel. It has L1 cache, 32KB data and 32KB instruction, for each physical core, L2 cache 256KB which is shared among logical cores, and L3 cache 8MB shared between all physical cores on the chip.

2.2. NVidia GeForce GTX 260

The emergence of programmable graphic hardware has led to increasing interest in off-loading numerically intensive computation to a Graphic Processing Unit (GPU). The combination of high-bandwidth memories and hardware that performs floating point arithmetic at significantly higher rates than conventional CPUs makes graphic processors attractive targets for a highly parallel numerical workload.

NVIDIA GeForce GTX 260 [7] GPU has a multicore architecture consisting of hundreds of cores. When parallelizing any algorithm, it should be decomposed into various parts, and each part is further broken down into tasks. Each task is executed on a different core of the GPU, and each task contains threads which are executed simultaneously. Each core has eight stream processors (SP). Each core runs in a Single Instruction Multiple Data (SIMD) manner in each clock cycle, all stream processors of a core execute the same instruction but operate on different data.

2.3. IBM Cell Broadband Engine Processor

The Cell BE is unique from traditional multi-core chips in that the nine cores that comprise the Cell BE aren't all functionally the same. One core, the Power Processing Element (PPE), is the master and runs typical PowerPC code. The other eight cores, the Synergistic Processing Elements (SPEs), can perform mathematical computation in parallel [9].

Each SPE unit has its own 256KB of memory that is loaded or stored from/to main memory via a Direct Memory Access (DMA) request to the Memory Transfer Engine (MTE). Once the local SPE memory transfer is complete, then the SPE is free to use local memory for calculations without any data or timing conflicts with other SPE units.

The Cell BE provides novel methods for parallelism. Each SPE uses Single Instruction Multiple Data (SIMD) vector registers to operate on multiple operands during one instruction. Operands may be 8, 16, 32, 64, or 128 bits wide depending on the computation needed. With 128 vector registers, each 128 bits wide, four 32-bit operands are available to perform four calculations simultaneously. The Cell BE includes DMA scheduling in hardware using priority levels and sequencing flags for automated queue management [9]. This allows the PPE to tell each SPE which of the eight DMA channels to use for its data transfers. All of the mathematic functions use packed vector registers whereby four simultaneous calculations are performed in one function call. The mailbox signaling technique will be used for communication between all six available SPE cores and the PPE core during execution of this algorithm. Computations in all six cores will be in parallel and in a pipeline fashion.

3. Parallel Implementation

3.1. Implementation on IBM Cell BE Processor

A Cooley-Turkey based FFT is used to convert the sensor array data from the time domain to the frequency domain [11]. Due to symmetry, only half of the resulting frequency bins are required as covariance input. For example, a 64-point FFT will result in 33 unique frequency bins. The FFT operation has been located inside the covariance module to maximize performance by eliminating unnecessary DMA transfers. An efficient bit reversal routine was used to perform the decimation-in-time swaps required. Vector registers are packed with data from the same row and column from four frequency bin matrices before being used by

the covariance routine. This is slightly different than the narrowband covariance module in that four covariance matrices are produced simultaneously instead of only one. To efficiently utilize the vector registers in the SPE units, the center frequency band is not used. For a 64-point FFT, this results in 32 frequency bins that map nicely to the four valued vector register format. So to compute the covariance for 32 frequency bins requires only eight separate matrix multiplications.

The MUSIC algorithm embedded in the wideband DOA algorithms [2][3] utilizes the fact that the signal vectors are orthogonal to the noise subspace. To generate the noise subspace, the covariance of the FFT data vectors of samples obtained from the linear antenna array is computed. The covariance matrix is computed as A'*A, where A is the FFT data vector samples and A' is the complex conjugate of A. The matrix A is in the complex domain so each element multiplication consists of $(a + bi) * (c + di) = (ac - bd) + (ad + bc)i$.

The covariance matrix is computed by taking each row and multiplying it with each of the other rows' complex conjugates, itself included as shown in Fig. 1. Each element of the row is duplicated into all four locations of a vector register. Multiplication is then performed on all four values simultaneously for four consecutive rows of the matrix. The multiplication result is then accumulated for each element of the matrix. This achieves the A'*A computation that results in the covariance matrix [2][3].

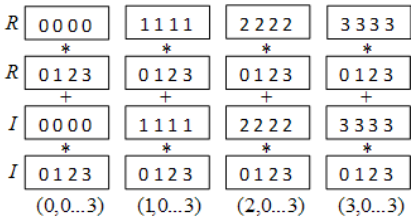

Figure 1. *Covariance calculation of real part for the first four matrix elements of A. The imaginary part is similar except instead of RRII, the order is RIIR and subtractions are performed instead of additions.*

The limitation of the SPE only being able to use local memory for calculation is handled by using mailboxes to issue memory pointers that point to the data to be acted upon. For example, the PPE sends in the mailbox slot for the covariance module, the memory pointer to the array sensor data and also the memory pointer of where to put the resulting covariance matrix. Subsequently, the covariance module returns a message back to the PPE indicating when the covariance computation is complete.

Using the square covariance matrix, eigenvalue-eigenvector decomposition is performed and the resulting eigenvalue-eigenvector pairs are sorted in descending order according to the eigenvalues. The eigenvalue-eigenvector decomposition is performed by taking the covariance matrix and mapping it to the real domain. If C = A + Bi, where C is the covariance matrix, then R = [A –B; B A], where A and B are matrices consisting only of the real and imaginary parts respectively. The resulting matrix R is both real and symmetric.

The real only symmetric covariance matrix is twice the width and height of the complex matrix it represents, however the complexity of the eigenvalue-eigenvector decomposition computation is significantly reduced. In addition, the eigenvalue-eigenvector decomposition method only uses the lower triangle given to it, therefore the –B values can be skipped and a simple duplication of the A values is all that is required. This rearrangement is performed inside the eigenvalue-eigenvector decomposition module to prevent the duplicated A values from being transferred from system memory twice.

To maximize the performance, all available cores need to be utilized at near 100%. In addition, the overall bandwidth achieved hinges on how many sensor array samples can be included in the covariance computation in a given amount of time. Therefore, as shown in Fig. 1, four SPE units were selected to perform covariance computations simultaneously, each on a separate block of sensor array data separated by time. The resulting four sets of 32 covariance matrices are then passed to the next cores for further processing.

The covariance matrix R is first reduced to tri-diagonal form using Householder transformations. The Householder transformation process is deterministic in nature and lends itself well to parallelization by vector register use. The next step is to reduce the tri-diagonal form to diagonal form using a combination of QL decomposition with implicit shifts and Givens rotations to maintain tri-diagonal form. The QL algorithm is not deterministic, so computation is terminated when convergence occurs within the limits of real number machine precision. Once the limit is reached, the resulting matrix is diagonal and represents the eigenvectors for matrix R.

Considerable effort was put forth into a parallel form of QL, however, the overhead complexity of tracking where in each matrix the Givens rotation should be performed, undermined the performance gain and introduced possible errors into the computation.In general, the diagonal decomposition converged in six iterations or less with the majority of iterations occurring in the upper half of the matrix.The lower half typically converged in one or two iterations.The convergence criteria was deemed true if the diagonal was within one epsilon unit of machine precision for single precision floating-point number representation. The result of the diagonal decomposition is a vector of eigenvalues with their respective normalized eigenvectors stored in matrix form. The eigenvalue-eigenvector decomposition is performed in the module labeled as EIG of Fig. 2.

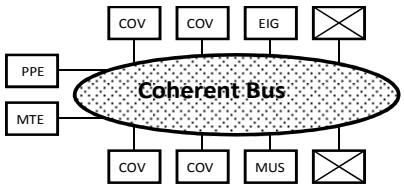

Figure 2. *Mapping of SPE modules.*

The last step of the eigenvalue-eigenvector decomposition is to sort the eigenvalues in descending order while keeping track of their respective eigenvectors. The eigenvector matrix is split into signal subspace and noise subspace using Akaike Information Criterion (AIC) [4]. The AIC criterion is used to determine the number of detected sources and separate the signal and noise spaces such that the number of signal eigenvectors matches the number of sources detected. The remaining noise space is then used as a guideline to detect the power peaks representing the detected signals and their respective angle of arrival. The number of signal sources is reported back to the PPE using the mailbox system. In addition, only the eigenvectors are returned by the eigenvalue-eigenvector decomposition module since the eigenvalues have served their purpose and are no longer needed.

The SPE unit labeled as MUS in Fig. 3 performs the computation of peaks using the power method, and lends itself well to parallelization. The power method is performed on the noise subspace eigenvectors for separate observations of the signals. The power is calculated as observed for each degree from 0 to 89 and returned back to shared system memory. The angles with the largest power peaks are the DOA of the incoming signals. It can be seen that all available cores of the Cell BE are all performing in parallel.

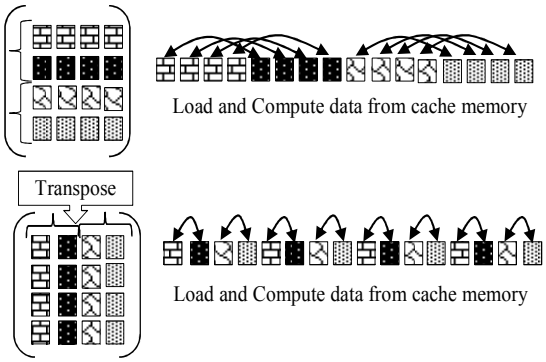

Figure 3. *Load and computer from cache with Row-Major technique.*

This initial DOA estimate, along with the remaining frequency bin covariance matrices, is put through a focusing transformation inside the 5b EIG module followed by eigenvalue-eigenvector decomposition. These eigenvalues and eigenvectors are again used in an AIC algorithm to estimate the number of sources. The number of detected sources is used to get the noise subspace. The power is calculated in 6b MUS as observed for each degree from 0 to 89 and returned back to shared system memory. The angles with the largest power peaks are the final DOA of the incoming signals.

3.2. Mapping Scheduling of DOA on Intel Quad Core and GPU

The wideband MUSIC algorithm can be divided into main computational blocks, namely computation of Fast Fourier Transform (FFT), covariance matrix, Householder Transformation, QR Decomposition, and Power Method. These individual blocks need to be partitioned, mapped, and scheduled on a Multi-Core/GPU so they can execute the entire algorithm efficiently.

The performance of parallel MUSIC algorithm is intimately related to the memory layout of the data. On modern shared-memory multiprocessors with multi-level memory hierarchies, sometime the column-major layout can produce unfavorable access patterns resulting in increased memory system overheads. These effects result in performance anomalies as signal size is varied.The traditional parallelization has restriction on the input data and required transform parameters. An Adaptive Load Balancing Algorithm is proposed for efficient implementation of MUIS with any size of signal on multi-core system.

3.2.1. Adaptive Load Balancing Algorithm (ALBA)

A task is a set of threads that executes on each core in parallel. In order to minimize the cache misses at each level of MUSIC, an optimal task size is very important. The Adaptive Load Balancing Algorithm (ALBA) creates different number of tasks at different number of task to balance the load. The execution of ALBA algorithm involves following three computational steps.

3.2.1.1. Row Major

The cache hit performance is improved by the concept of row-major organization [9]. The element of the signal is accessed column wise, and therefore is not in sequential order in memory. This is due to the fact that signal is stored in memory as row-major order. As a result, the MUSIC algorithm is bandwidth limited and displays poor performance and low efficiency because of time spent in loading data rather than computing when performing computation. In order to optimize the performance of columns wise the signal is transposed and then the computation is performed on each row instead of columns wise as shown in Fig. 3.

3.2.1.2. Data Level Parallelism

MUSIC algorithm is modified for execution on multicore, especially when dealing with the cache-memory. With data level parallelism, the goal is to maximize the utilization of cache memory for each core. It is of great importance to use memory resources efficiently, as it is apparent that the maximum performance obtainable from current multicores is mostly limited by the memory access.

A review of the profiler indicated that if installed RAM in a computer system is between 2GB to 4GB then 40% of L1 cache is consumed by the Operating System (OS) of each core.

To find the maximum number of data points that the L1 cache may hold can be calculated as.

$$Max\ (C)\ = \frac{(C_{L1} - C_{L1} * 0.4)}{D} \qquad (1)$$

Where

- C_{L1} is the total L1 cache (32KB)
- $C_{L1} * 0.4$ is cache used by operating system

- D is number of bytes of each data point
- Max (C) is maximum data points to be stored into available L1 cache.

Therefore the maximum data points which can be stored in L1 cache would be Max (C) samples. In this way, each sub problem should not be more than Max (C) samples to get the maximum performance of L1 cache of each core. Eqn.2 and 3 are used to calculate the total number of bytes for Max (C) data points and total number of tasks to execute each task independently on each core.

$$\text{Mem}(\text{ Bytes}) = \text{D} * (\text{L} + 1) \quad (2)$$

Where

- D is number of bytes of each data point
- L is the total number ofdata points
- Mem is the total bytes for all data points
- 1 is considered as an extra element as OS required four bytesforcreation of an instance of an object

$$L_T = \frac{x(n)}{Max\ (C)} \quad (3)$$

Where $L_T$is total number of tasks

If the number of samples is less than or equal toMax (C), then entire signal is considered as single task. Otherwise the signal is divided into n sub-signals, and then each sub-signal is considering as a task as shown in the following pseudo code. In this work, it has been found that most of the time, the data is greater than Max (C) samples.

Creating Tasks Pseudo Code

1. Input signal x(n)
2. Data store into array
3. $Max\ (C) = \frac{C_{L1} - C_{L1} * 0.4}{D}$
4. $Mem(\ Bytes) = D * (L + 1)$
5. $L_T = \frac{x(n)}{Max\ (C)}$
6. $If L_T \leq Max(C)$

then $Task = 1$

else

i. $x = \frac{x(n)}{Max\ (C)}$

ii. $y = x - round(x)$

iii. $if y \leq 0.5$

iv. $n = round(x)$

$else$

v. $n = x$

1. $for\ 1\ to n$
2. $Task_i = x$ Where i=1,2…..n
3. $Task_{n+1} = \left|(n - x) * round(x)\right|$

3.2.1.3. Thread/Operation Level Parallelism

Thread level parallelism is a form of parallelization that is used across multiple cores. The processor used in this study has four physical cores and each physical core has two virtual processors running in parallel. Tasks are assigned to the physical cores and threads are assigned to the virtual processors as shown in Fig 3.

The process of thread level Parallelism is described as follows.

- Data points are converted into n tasks, depending upon available L1 cache.
- Tasks are assigned to each physical core. Each task contains multiple threads.
- Threads are assigned to virtual cores as each physical core contains two virtual cores.

3.2.2. Parallel FFT

Starting with the initial parallel implementation of the FFT [12][13], followed by analyzing the problem, a sequence of optimization techniques are used to improve the performance of the initial parallel implementation. In initial implementation, sample data is divided into different tasks, tasks are created on based of ALBA. Each task contains 64 data points. These tasks are sent to global memory of GPU and global queue of multi-core then to share memory, because all SPs of GPU and all cores of multi-core share the data from shared memory and the data can be used among multiple threads across the task for GPU. Each task is divided into subtasks. Each subtask computes a butterfly operation which includes reading of 2-point data, twiddle factor from the pre-computed array, performs butterfly operations on them and writes back to the cache (L1) for multicore, shared memory for GPU, and synchronized.

By using 2-point subtask, a 64-point FFT computation can be completed with 32 subtasks. If four data points are taken at a time instead of taking two points at a time, then two butterfly operations can be computed independently. The thread can then read all data from cache of multicore or from shared memory of GPU and write back the results. This reduces the total number of memory access operations. Consideration of four data points at a time will require synchronization after two butterfly operations. This will eliminate half of the synchronization operations. Decreasing the number of synchronizations and memory operations can potentially improve the performance. Since the system can handle computations of up to eight data points at a time due to cache limitations, then this will result in 100% improvement in the computation time.

In a 64-point FFT (32 butterfly operations), 32 twiddle factors are used in the first stage, half of those twiddle factors used in the second stage are the same as the values used in first stage, and so on. Twiddle factors are stored in cache thus reducing unnecessary memory accesses. This will provide further improvement of 10.2%.

3.2.3. Parallel Covariance Matrix

The initial implementation of covariance matrix is accomplished via blocking and row-major techniques [14]. Consider an example of a 4x4 matrix, in initial implementation for blocking; four sub-matrices (blocks) are created. The number of columns in the matrix corresponds to the number of sensors, in case of four sub-matrices; the number of sensors is divided into two parts, which have to combine back when synchronizing the tasks, which is an expensive operation.

To optimize the performance, two sub-matrices of 2x4 are

created, which have reduced 50% synchronization overhead and do not have to combine the sensors back at the time of synchronizing the tasks. This improved the performance by 50% from initial implementation. Each sub-matrix is a task. These tasks are sent to global queue of multicore and global memory of GPU.

3.2.4. Parallel Householder Transformation

Householder transformation and QR decompositions are used to compute eigenvalues and eigenvectors of the noise subspace [4]. Householder transformation is an efficient and popular technique to transform covariance matrix into a tri-diagonal matrix. The QR decomposition method uses tri-diagonal matrix and produces eigenvalues and eigenvectors.

Householder transformations are partitioned into seven steps to calculate the tri-diagonal matrix as shown in Table 1. Partitioning this part of the algorithm consists of a combination of steps requiring sequential processing (Steps 1-3) and parallel processing steps (Steps 4-7). The steps 4-7 are matrix multiplication as shown in Table 1. These steps can be performed similar to the method used in the covariance matrix section.

Table 1. *Householder Transformation.*

1. S= Sign $[X_k]$ $\sqrt{\sum_{j=k-1}^{n}(X_j)^2}$	Sequential Processing
2. R = $\sqrt{2\,S(S+x_2)}$	
3. W = $\frac{1}{R}\{0, \dots\dots, 0, S+x_2, x_3 \dots\dots x_n\}$	
4. V = A * Transpose(W)	
5. C = W * V	Parallel Processing
6. Q = V – (Transpose(W) * C)	
7. Values = A -2.0*(Q*W) – Transpose(2.0-(Q*W))	

3.2.5. Parallel QR Decomposition

For a symmetric tri-diagonal matrix, the QR decomposition is an efficient way to obtain the eigenvalues. The algorithm is based on the fact that such a matrix can be factored into Q and R, where Q is an orthogonal transformation and R is an upper right triangular matrix. QR decomposition is partitioned into six steps for calculation of the QR matrices. Partitioning of QR algorithm consists of a combination of steps requiring sequential (due to data dependencies) and parallel processing. A parallelized QR decomposition algorithm is shown in Table 2. It can be seen from Table 2 that steps one through three are sequential. Steps four to six are executed in parallel in similar fashion as explained in parallel covariance matrix part.

Table 2. *QR Decomposition.*

1.S= Sign $[X_k]$ $\sqrt{\sum_{j=k}^{n}(X_j)^2}$	
2.R = $\sqrt{2\,S^2 + 2S(x_k)}$	Sequential Processing
3.W = $\frac{1}{R}\left\{0, \dots\dots, 0, \frac{X_k+\text{sign}(X_k)S}{S} + \frac{X_{k+1}}{S}, \dots\dots \frac{X_n}{S}\right\}$	
4.H^1 = 1-2W* Transpose(W)	
5.Q = H^1 * H^2.....H^n	Parallel Processing
6.R = (H^n * H^{n-1}.....H^1) * A	

Parallel AIC and Power Method

The Akaike Information Criterion (AIC) [4] is used to determine the model order. It helps us in separating the signal and noise subspaces. AIC is divided into (number of sensors -1) tasks, subtasks, and threads and they have been mapped onto both the Multicore and GPU.

Finally the Power Method starts with an arbitrary vector, which may be an approximation to the dominant eigenvector. QR decomposition produces distinct eigenvalues λ1, λ2, λ3… λn and they are sorted in descending order. An eigenvector V1 corresponding to λ1 is a dominant eigenvector. Power Method is divided into two steps and both steps can be executed in parallel. Each step contains multiple tasks, subtasks, and threads, and they have been mapped onto both the Multicore and GPU.

4. Simulation

In order to demonstrate the parallel implementation of the DOA algorithm for wideband signals, a uniform linear array of sixteen equally spaced Omni-directional sensors was used. Two wideband sources atθ_1 and θ_2 were assumed. The signals are stationary zero mean band pass white Gaussian processes. Simulation data is similar to the method described by Wang &Kaveh [3]. The ease of signal detection using wideband DOA on a sixteen element sensor array is clear in Fig. 5. Four different FFT point sizes were tested and compared. The 64-point and 32-point FFTs show approximately the same results. Even the 16-point FFT gives adequate signal DOA detection, however the 8-point FFT starts to show extra peaks that could be misconstrued as valid signal sources. Computation times for the building blocks of covariance matrix, Householder transformation, QR decomposition, AIC, and Power Method are shown in Table 3. These computational blocks have been parallelized and mapped on Multicore and GPU. The parallelized wideband CSS algorithm has been implemented on Multicore and GPU. It has also been simulated on single core for comparison purposes. Simulation results of wideband CSS based DOA MUSIC algorithm with sixteen sensors and two sources at 20o and 50o are plotted in Fig. 4.

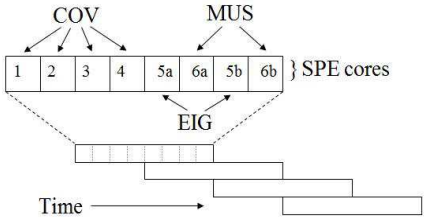

Figure 5. *Pipeline stages of wideband DOA. The focusing is performed inside the 5b EIG after getting the initial DOA estimate from 6a MUS.*

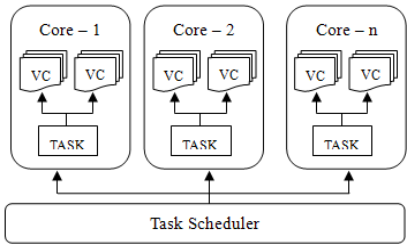

Figure 4. *Thread level Parallelism.*

Table 3 shows the computation time achieved using various approaches to the covariance module. The covariance computation time determines the number of possible sensor array samples processed over a given time and therefore the overall bandwidth achievable by the DOA algorithms. A simple covariance calculation using single-threaded code running on the PPE can only provide a bandwidth of 11 KHz. Even though the Cell BE runs at 3.2 GHz, the resulting bandwidth is very narrow. Parallel narrowband MUSIC when configured to utilize all six available cores has a respectable bandwidth of approximate 500 KHz. Parallel narrowband MUSIC with an unrolled inner loop can improve bandwidth by 50 KHz. This is done simply by unrolling the inner covariance multiplication loop, thereby leveraging many of the 128 vector registers available in the SPE unit. Parallel wideband DOA using various FFT transform lengths were computed and their bandwidth data is also given in Table 3. The 16-point FFT size demonstrates a good compromise between DOA resolution and bandwidth performance. It can be seen from Table 3.

Table 3.*Computation Time of MUIC.*

	Single , Quad Core and GPU			**IBM Cell Processor**	
	Single core	**Nehalem Quad Core**	**GPU**	**Single Thread**	**Multi-Thread**
FFT	1.283	0.589	0.394	102.208	5.783
Covariance	0.508	0.144	0.135		
Householder	0.273	0.122	0.119	12.791	6.294
QR	0.230	0.118	0.113		
AIC	0.180	0.080	0.075	1.724	4.005
Power Method	0.150	0.068	0.066		
Total	2.624	1.121	0.902	116.723	16.082
% Improved		3.13	3.74		7.25

5. Conclusion

Computation of Direction of Arrival (DOA) via MUSIC algorithm has been parallelized and implemented on Intel Nehalem Quad Core, NVidia's GeForce GTX 260 GPU, and IBM Cell BE Processor. It concludes that computation of this algorithm is faster with NVidia's GPU as compared to Intel Nehalem Quad Core and IBM Cell BE Processor. It was also determined that computation of an algorithm with large data sizes will perform better than one with smaller data sizes. This is due to the balance between communication overhead and computation time. A large number of tasks with smaller data sizes will have a long computation time as compared to a small number of tasks with large data sizes, because too many tasks add overhead and too few tasks leave processors idle. Therefore, there needs to be a balance between data size and the number of tasks.

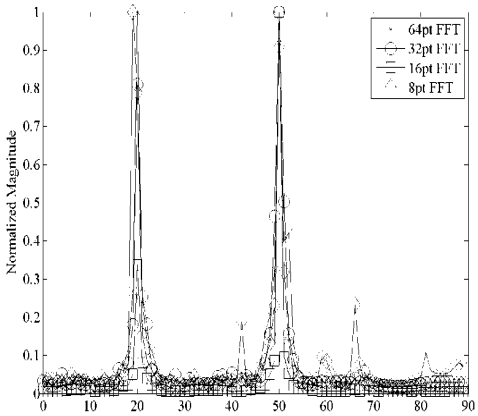

Figure 6. *Wideband DOA of two incoming sources from 20 and 50 degrees using 4096 sensor array samples of a sixteen element array with four different FFT sizes.*

References

[1] Joohn L. Hennessy, David A. Patterson “Computer Architecture a Quantitative Approach” Morgan Kaufman Publishers 2008.

[2] R. O. Schmidt, “Multiple emitter location and signal parameter estimation” IEEE Transactions on Antennas and Propagation, vol. AP-34, No. 3, pp. 276-280, March 1986.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford. Clarendon, 1892, pp.68–73.

[3] H. Wang, M. Kaveh, ”Coherent Signal-Subspace Processing for the Detection and Estimation of Angles of Arrival of Multiple Wideband Sources,” IEEE Transactions on Acoustic, Speech and Signal Processing, Vol –ASSP-33, No. 4, August 1985, pp 823-831.

[4] Akaike, H. ,”A New Look at the Statistical Model Identification,”IEEE Transactions Automatic Control, Vol. AC-19, pp. 716-723, December 1974.

[5] Intel Corporation.Intel Nehalem

http.//www.intel.com/technology/architecture-silicon/next-gen

[6] Nvidia Corporation Geforce GTX 260

http.//www.nvidia.com/object/product_geforce_gtx_260_us.html

[7] IBM, Software development kit for multi-core acceleration version 3.1. Programmer’s guide, Retrieved from

http.//publib.boulder.ibm.com/infocenter/systems/topic/eicct/prg

[8] J. Bartlett, Programming high-performance applications on the Cell BE processor, Retrieved from http.//www-128.ibm.com/developerworks/power/library, 2007.

[9] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, Numerical recipes. The art of scientific computing, 3rd ed., Hong Kong. Golden Cup, 2007.

[10] Naga K. Govindaraju, Brandon Lloyd, Yuri Dotsenko. “High Performance Discrete Fourier Transforms on Graphics Processors”, Microsoft Corporation.

[11] William J. Pilaud. “Improved FFTW Benchmark to Measure Multi-Core Processor Performance”, Curtis Wright Controls Embedded Computing.

[12] M. S. Lam, E. E. Rothberg, and M. E. Wolf. “The cache performance and optimizations of blocked algorithms”. In Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems, pages 63.74, April 1991.