FFT PARALLEL IMPLEMENTATION FOR MRI IMAGE RECONSTRUCTION

Andrei ȚUGUI[1]

Această lucrare prezintă implementarea algoritmului FFT (Cooley-Tukey) - intens utilizat la reconstrucția imaginilor RMN - pe o maşină revoluționară de calcul paralel, Connex Array. Valorificând structura şi prelucrarea vectorială a informațiilor in Connex Array, reconstrucția unei imagini RMN s-a facut mult mai rapid decat în majoritatea scanerelor comerciale. Rezultatele sunt remarcabile. Propunem cu această lucrare utilizarea Connex Array în sistemele RMN în timp real, unde timpul de reconstrucție al imaginilor creşte vertiginos odată cu creşterea numărului de secțiuni de explorat.

This paper describes FFT Cooley-Tukey algorithm implementation used in MRI image reconstruction on a revolutionary parallel computing machine, Connex Array. By taking advantage of it's vectorial structure and processing manner, MRI image reconstruction was much faster than most of usual MRI commercial scanners. Results are remarkable. Our proposal in this paper is the use of Connex Array in real-time MRI systems, where reconstruction time grows seriously as the number of taken sections grows.

Keywords: FFT, Connex Array, MRI, parallel computing

1. Introduction

FFT (Fast Fourier Transform) is a fast algorithm for Discrete Fourier Transform (DFT) computation. In the literature, [1, 2] FFT has been extensively studied and implemented as an important frequency analysis tool in many areas such as image processing, signal processing and other domains. There are many variants of the FFT algorithms. In this paper, we focus on the most common FFT algorithm, the radix-2 Cooley-Tukey algorithm [3] used in magnetic resonance image reconstruction (Fig. 1).

Magnetic Resonance Imaging (MRI) [4] uses magnetic fields and radiofrequency pulses to view different kind of organs and tissues. Often, MRI offers information can not be seen using X-Ray, ultrasound or CT (Computed Tomography) scan [5]. We mention projection reconstruction used in MRI being commonly used by other medical imaging techniques like CT or PET [6, 7, 8, 9], very simple and fast technique for image reconstruction [10] [11].

[1] PhD student, Electronics, Telecommunication and Information Technology Faculty, POLITEHNICA University Bucureşti, România. e-mail: andrei.tugui@yahoo.com

In this paper we will show that using a *green* and cheap parallel-computing technology (Connex Array), the main actual MRI computational drawback (high computational time) [12] can be eliminated. Also, [13, 14] describe several actual software platform/implementation for CT and PET Fourier-based image reconstruction claiming high reconstruction time, even though parallel-processing techniques are approached. Other parallel approaches in actual medical imaging are presented in [15].

1.1. K-space and FFT image reconstruction

In MRI acquisition process, sample FID (Free Induction Decay) data is brought into the K-space (Fourier related space) by any of the linear or non-linear methods known [12]. The most used MRI scanning methods available use linear filling methods in K-space because FFT is suitable for fast data reconstruction. [12]

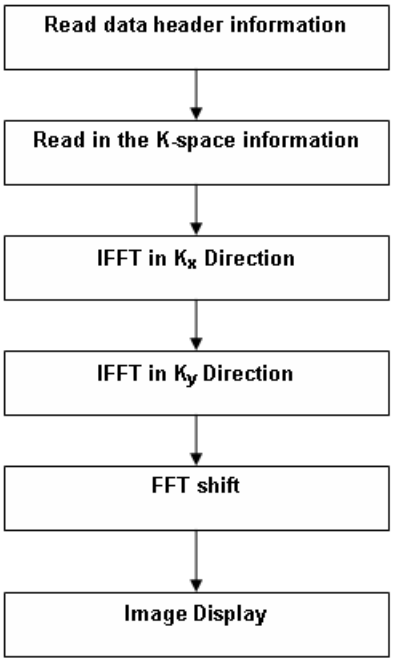

Fig. 1. MRI reconstruction pipeline

We will show in this paper that FFT algorithm (Cooley-Tukey) [3], commonly used to reconstruct a MRI image (taking about 80% of image reconstruction time [12]), is very suitable for our parallel-computing machine,

Connex Array. Moreover, using its parallel computing power, real-time MRI will be seriously improved.

FFT technique involves some Fourier series on sample linear data or polar data. [10]

As can be seen in Fig. 1, Discrete 2D Fourier Transform of an image means computing one 1D FFT (One Dimension FFT) on the input data line, and then computing one other 1D FFT on every column resulted from the first FFT partial data from K-space matrix (input-output operation as reading or shift data (Fig. 1) can be easily neglected as they take few cycles in computation using our parallel architecture [7]).[17] The same result (the image) is achieved no matter the computing order line-column.

Although today we have rapid processors, high dimension image reconstruction (512 or 1024) and high number of receiver channels (8…16) make a very hard mission for processing machines (Table 1).

Table 1

MRI performances on different processors

Processor	Function	Image dim. (pixels)	Computational time (s)
SGI R10000 175 MHz	3D TFD Rotation	128x128 x30	1.5
SGI R10000 175 MHz	3D TFD Rotation	256x256x30	27
HP PA-8000 200MHz	3D TFD Rotation	128x128x30	1.3
HP PA-8000 200MHz	3D TFD Rotation	256x256x30	24.7
Pentium II 400 MHz	3D TFD Rotation	128x128x30	0.8
Pentium II 400 MHz	3D TFD Rotation	256x256x30	13.8
SGI R10000 175 MHz	3D TFD Reconstruction	80 images, 128x128x30	335
HP PA-8000 200MHz	3D TFD Reconstruction	80 images, 128x128x30	276
Pentium II 400 MHz	3D TFD Reconstruction	80 images, 128x128x30	162.1

2. Connex Array

This paper machine's architecture belongs to a revolutionary project of parallel computing age, „*The Connex Project*".

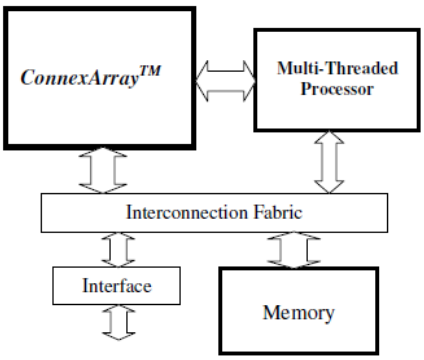

Fig. 2. Connex Array

Connex Array is a parallel computing machine which best fits into "*Terra Architecture"* concept [7], a high performance architecture including several types of parallelism to optimize chip's area and power consumption (*P = 2.5 W*, $f_{clk} = 400MHz$).

Connex Array's hardware structure contains 1024 processing elements (PE) ring connected, each having 256 registers (meaning we have a 256 rows by 1024 columns matrix). ConnexArray^{TM} is a many-cell array of execution units (EU) designed for intense computations, while MTP (Multi-Threaded Processor) is a multi-core BEAM (Bubble Free Embedded Architecture for Multithreading) processor for complex computations, dealing with both scalar and vector-specific instructions.

The execution model involves:

- in each clock cycle an instruction sequencer (IS) broadcasts one instruction to be executed by each EU;
- each EU executes the received instruction according to its internal state (stated by the selected Boolean), for example:

where (bool_vect_q == 1)
vector_n = f(vector_m, vector_p);
elsewhere
vector_n = g(vector_m, vector_p);

- the instruction operates on data stored in each EU and, sometimes, on some data stored in a small neighborhood (usually in EU_{i-1} and EU_{i+1});
- the sequence of instructions evolves according to the IS internal state and according to the scalars or Booleans provided by the reduction tree. (Fig. 3)

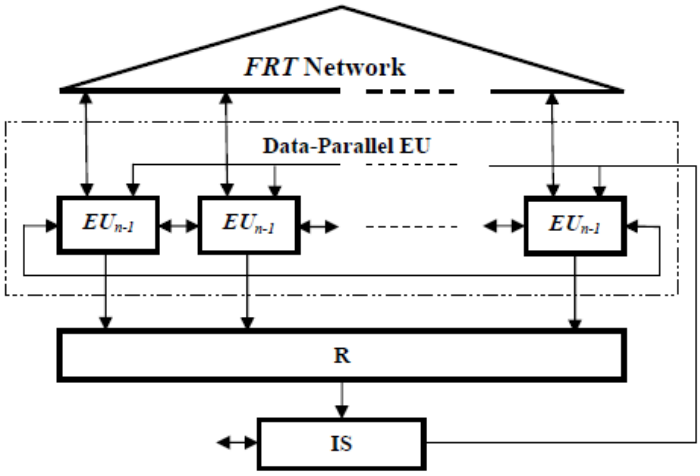

Fig. 3. CA 1024 many-processor architecture

On 65 nm technology, our architecture can be implemented on a 2.5 cm^2 silicon area [7]. We used Connex to reconstruct an usual 128x128 MRI image. We compared Connex to other parallel chips performance and power consumed for the same algorithm implementation. Connex's detailed description can be found in [16, 18]. Only the components of m, n and p vectors selected by the boolean vector q are involved in the execution (Fig.4)

Connex Array is programmed in VectorC [19], a C++ language extension working with new primitives (*vectors*) like: int vector (array of integer data types), float vector (array of real data types), etc. By *vector* we mean a new N-length data type containing K-space complex pixels modelling the vectors in Connex Array. A statement like:

$$v_3 = v_1 + v_2 \tag{1}$$

replaces following sequential programming code:

for (int i = 0; i < SIZEOF_VECTOR; i++)
$\{v_3[i] = v_1[i] + v_2[i]\}$

where v_1, v_2, v_3 are float vectors and *SIZEOF_VECTOR* variable holds vector's length [19].

In this paper *SIZEOF_VECTOR* = 128, since we compute a 128x128 K-space matrix.

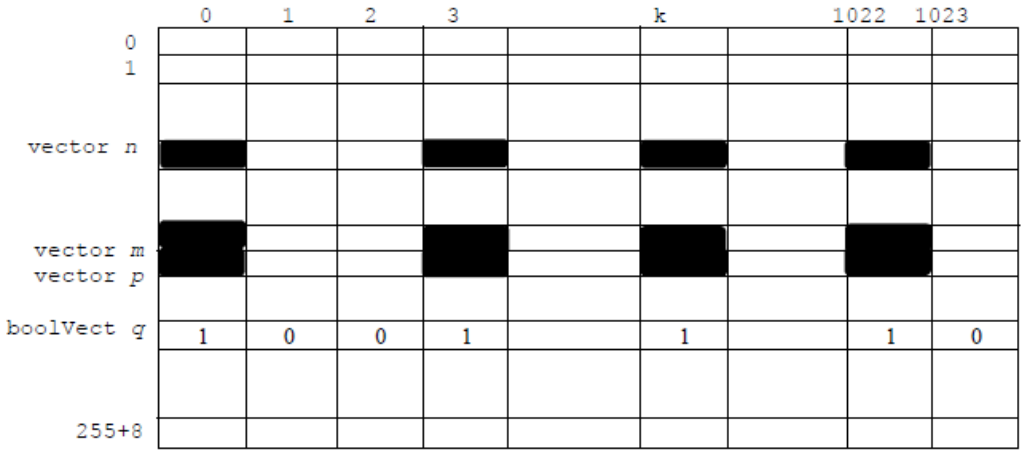

Fig. 4. CA 1024 structure (256 scalar vectors and 8 bool vectors)

Assuming we have a vector of integers vector int v, to fill values "0" in even positions and "1" in odd positions we declare [19]:

where (v%2 == 0)
{v = 0;}
elsewhere
{v = 1;}

Connex paralelism cam be simply described if we refer to its "full line operation" feature, meaning an arithmetic operation performed having generic form like $R = A * B$, where R is the result and A, B the operands, replaces $R = \sum_{k=0}^{N-1} A_k * B_k$. This is what we call vectorial computation. Vectors like A, B can be declared as follows:
vector A,B;

3. FFT and Connex Array

FFT became a very used algorithm in scientific and engineering applications; it is concerned about following aspects:

- data access, step by step;
- sinus (cosine) function computation;
- computational precision.

To compute a sample F(k) for FFT (Fig. 5), N complex multiplications and (N-1) complex additions are necessary [17]. Therefore, the total number of operations necessary for the entire image reconstruction is:

$$N * N = N^2 \text{ complex multiplications,} \quad (2)$$

$$(N-1)N = (N^2 - N) \text{ complex additions} \quad (3)$$

Our FFT data amount was 168668816 floating point operations, computed in 0.67 ms (*Table 3*).

4. FFT algorithm implementation for image reconstruction. Comparation to NVIDIA

The image reconstruction technique requires 1D FFT computation horizontally on sample data arranged in 128 length vectors, then computing another 1D FFT but vertically, on the same data set. [17]

To compute FFT Cooley-Tukey algorithm (Fig. 5) with Connex we used VectorC library [19], which by C++ operator's overloading models basically *vectors*. Our 2D FFT Cooley-Tukey algorithm requires the computation of one $N = N_1 * N_2$ FFT using twiddle factor multiplications between vertical and horizontal stages:

- N vertical N size FFTs;

- Array multiplication with twiddle factors $e^{\frac{i2\pi}{i}(NM/MN)}$
- N vertical N size FFTs.

One MRI image reconstruction 128x128 pixels, 32 bits floating point was made, using Connex Array. Benchmark results can be seen in Table 3.

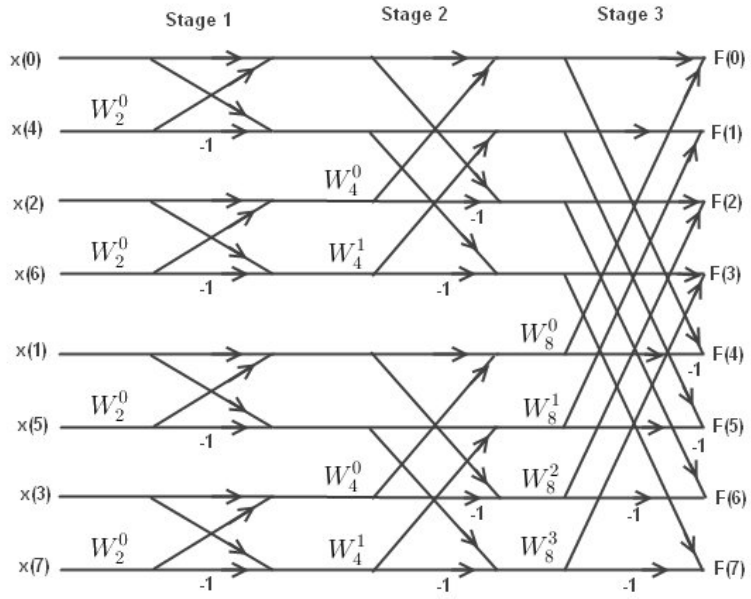

Fig. 5. FFT Cooley-Tukey algorithm, decimation-in-time (N = 8)

FFT algorithm (Cooley-Tukey) for N = 8 is described in [3]. For N = 128, FFT computation takes N = 7 stages ($2^7 = 128$) (Fig. 5).

Main operations [19] involved in our FFT parallel algorithm are:

- vector *addition:*

$$v_3 = v_1 + v_2 \tag{4}$$

- vector *multiplication:*

$$v_3 = v_1 v_2 \tag{5}$$

- *conditional execution:*

$$v_3 = b_3 \,?\, v_1 : v_2 \tag{6}$$

$$v_2 = v_1 << n \tag{7}$$

where n is the bit number shift for v_1 and b_3 is a bool value.

Sample vector computation each stage (Fig. 3) is made by adding the old value of the sample with the value resulted from the current step, as follows:

$$X[i] = X[i-1] + (x_{11} + c_1 x_{12}) \tag{8}$$

Intermediate vectors like x_{11}, x_{12} were computed using shift functions detailed in paper [16], as follows:

```
i = 0;
temp = shiftLeft(X[0],k);                                   (9)
where (INDEX < k) { x12 = temp;}
i = i + k;
temp = shiftRight(X[0],k);
where (INDEX >= i && INDEX < (i+k)) { x12 = temp;}
```

Here, INDEX is a particular vector filled in with consecutive scalars from 1 to *SIZEOF_VECTOR*.

We used C/C++ working environment using a special class (*vector*) dedicated to Connex Array applications [19]. All operations made were floating point, because real samples achieved by scanners are complex [12], as follows:

$$x(i) = \mathrm{Re}\{x(i)\} + i * \mathrm{Im}\{x(i)\} \tag{10}$$

Coefficient vectors ($c_1 \ldots c_7$) filled in with twiddle factors [3] on every computational step are defined as follows:

```
where (INDEX%2 == 0) { c1 = 1;}
elsewhere { c1 = -1;} // c1 = [1 -1  1 -1  1 -1...]
```

For computational simplicity, we considered $w_{128}^0 = w_{128}^1 = \ldots w_{128}^7 = 1$ (twiddle factors) [17, 3], and all vector samples $x(i) = 1 + i$.

As workbench, we used Eclipse IDE for C/C++ Developers © [24]. Due to Connex's parallel structure, 1024 samples properly arranged (Fig. 9) can be computed in a single clock cycle [16, 20]. One floating point operation takes 16 clock cycles [7].

As first landmark we consider Table 2 conclusions from paper [20] concerning 1D FFT computed with Connex:

Table 2

1D FFT, with reordering. N = FFT' s dimension , M = number of FFT, 32 bit floating point

N	M	Cycles	Cycle/FFT	Mops
4	256	142	0.6	14422.5
16	64	562	8.8	7288.3
64	16	1342	83.9	4578.2
256	4	3562	890.5	2299.8
1024	1	7447	7447.0	1375.1

From hardware-algorithmic point of view, Connex is an array counting P = 1024 columns and M = 256 rows, with N <= P. Thus, we can refer the "i" register from "j" processing element [7] as $A[i][j]$. (Fig. 4)

Our FFT reconstruction algorithm used 2 Radix-2 IFFTs "Decimation-In-Time". Both vertical and horizontal FFTs represent the paralelisation of FFT serial code described as follows:

For the horizontal step, N = 128 FFTs. We declared all 128 vectors identical:

$$X[k] = INDEX, \quad k = \overline{0,127}$$

Buffer vectors for butterfly operations each stage were declared as follows:

$$\text{vector } temp, x_{11}, x_{12};$$

Buffer vector holding first horizontal FFT is vector buffer [128]. First FFT is computed for all $X[k]$ vectors in $n = 7$ steps. The serial code parallelized with Connex involves:

```
for (i=1; i<=N; i++)
  {
    for (j=1; j<=n; j++)
      {
            for (k=1; k<=N; k++)
              {
                butterfly_vert(i)
              }
                        X[j] += butterfly_vert(i)
      }
  },                                          (11)
```

where N = 128, n = 7 and butterfly_vert(i) = $x_{11}[i] + c_j * x_{12}[i]$.

Using Connex, all loops are parallelized at once. Due to butterfly model, all vectors must be shifted horizontally (Fig. 6)

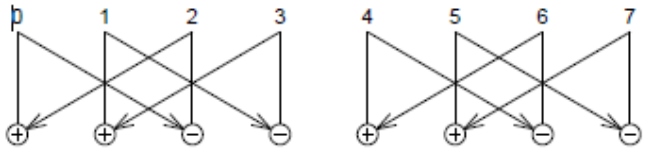

Fig. 6. Radix 2 FFT, stage 4: the groups can be extended horizontally until all PE are filled

We have $\log_2 N$ stages, each requiring *n* twiddle factors; that means $n\log_2 N$ coefficients filling *n* scalar vectors that can be loaded once in Connex array space from external memory and used forward to other computation. All shifting operations are shown in Table 3.

Table 3

Shift weight for 1D FFT, N = 128

Stage	Shift number (left and right)	Shift positions
1	2	1
2	128	2
3	128	4
4	64	8
5	32	16
6	6	32
7	1	64

Each stage has N butterfly_vert() operations involving N/2 additions and N/2 complex multiplications; algorithm in (11) repeats twice identically to compute the imaginary parts FFT, $\mathrm{Im}\{X[k]\}$. Knowing that for 2D IFFT (one image reconstruction) N^2 multiplications and $(N^2 - N)$ complex additions are involved, also that every floating point operation requires 16 clock cycles and we counted 359 shifting operation (Table 3), total parallel execution time results:

$$T_{2DFFT}(N) = 16 * N^2(1 - N) + y\,, \tag{12}$$

where N = 128 (FFT size) and y = 359 (total number of shifting operations).

As for I/O throughput time, we can briefly summarize the total time elapsed to load all $2N^2$ data samples from external memory into the array:

$$T_{i/o} = 2N^2 C_{i/o}\,, \tag{13}$$

where $C_{i/o}$ is average time required to load a sample data from external memory into data array (K-space). $C_{i/o}$ can easily be neglected, as I/O operations are transparent to the user [7]; but as N grows, shifting operations become the dominant computation factor. Still, having 1024 PE, N <= 1024 will be the chip's restriction limit. All data can be buffered in 4 M vectors in Connex (where M = 256). Still, if N < P we can compute P/N 2D FFTs in parallel. Ex: 16 complex 64x64 FFTs can be computed in parallel by splitting the data array organized in 1024 columns by 256 rows into 16 blocks 64x64 samples each.

As it is showed in paper [20], vertical FFT computation (this paper approach) is the most efficient for quadratic FFT dimension like ours.

We will show in the following a comparison of FFT parallel implementations on NVIDIA Quadro FX, a powerful landmark in graphic parallel processing (*Table 6*), and Connex Array. Although NVIDIA acts like a strong parallel computing machine (486 Gflops general performance and Connex only 117 Gflops), FFT 2D algorithm implementation on Connex is 8 times more efficient (*Table 6*).

On NVIDIA Quadro FX, each of 8 computational stages (Fig. 5) require distinct program fragments (multipass algorithm). WGL_ARB_pbuffer is a I/O buffer counting several draw buffers used in one stage. WGL_ARB_render_texture is another buffer used to save data output from each stage and to load input data into the next stage. For 32-bit float I/O samples NVIDIA uses a dedicated buffer also, NV_float_buffer. By creating 8 draw buffers (ATI_draw_buffers), 2 FFTs can be computed in parallel using a single I/O WGL_ARB_pbuffer, taking as input 4 samples and then the output is redirected as input to the next stage. In the following we present 2D FFT CUDA (Compute Unified Device Architecture) parallel algorithm used by NVIDIA:

- first is created one I/O (Input-Output) buffer GL_FLOAT_R32_NV, counting 8 32-bit scalar buffers (GL_FRONT_LEFT… GL_AVX3); [23]

- at first step, the first 4 scalar buffers are used as load source for draw buffers, and the last 4 as destination. Two scalar buffers store real parts for the first FFT, and other 2 scalar buffers store imaginary parts. The rest 4 buffers store real/imaginary parts for the second FFT;

- at next steps, draw source and destination buffers are switched eachother, and the process continues. After completion of all stages, remaining data from draw destination buffers are filled in with computed 2D FFT.

Quadro FX architecture uses one “fragment processor”, and two dedicated complementary graphic processors: “vertex” and “rasterizer” [23]. Two approaches are presented in paper [23] for our 2D FFT parallel implementation: one is called “Mostly loading the fragment processor”, where all K-space sample data is loaded in parallel into the fragment processor, 1D “butterfly lookup” textures for data mixture twiddle factors computation are created, and a fragment code program is executed for each K-space sample; but this approach makes from fragment processor a bottleneck, so vertex and rasterized processors are idle for a long time. Second parallel approach is called “Load the vertex processor, the rasterizer and the fragment processor”: here specific quads are created for each fragment code (Fig. 7), so lot of quads for early stages and few for final ones, process requiring adaptive load into vertex, rasterizer and fragment processor, making the entire algorithm more complex. Twiddle factors computation is made with a dedicated function (ARB_fragment_program). For each stage, fragment

groups are joined into compact blocks (like those coloured in blue and green in Fig. 7), and then passed to the vertex processor for index reordering and sign computation; moreover, load balancing is passed also to vertex and rasterizer.

Using Connex, all data computation means flexible data load and basic operations with *vectors*. For both FFTs, 128 vectors representing K-space data were loaded from I/O Plan [7] into memory in one single clock cycle (Fig. 8), also coefficient vectors (twiddle factors) which are not computed anymore unlike NVIDIA's algorithm. Each stage data results are computed using a vector buffer *temp* (9) and precomputed coefficient vectors (8), (9). Unlike NVIDIA, there is no need for switching buffers, since *temp* and X[i] are overwrited each stage; also, vector-oriented approach specific to Connex eliminates the vertex and rasterized processor necessity from NVIDIA proposed chip, as we can directly compute up to 256 K-space rows using the MTP [7]. Other NVIDIA algorithm limitations we can mention are: it involves several syncronization between the nuclei asociated to each thread, data must be copied from primary memory into video dedicated memory and back and there are some deviations from IEEE 754 standard concerning floating-points support. Also, CUDA recommandation for high number of threads require computing 32 groups once for best performance.

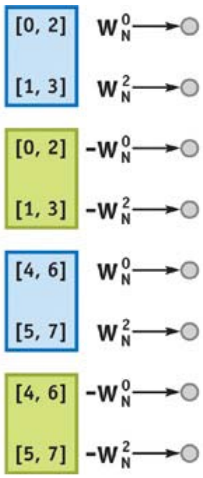

Fig. 7. Decimation in time FFT butterfly (stage 2, N=8) with NVIDIA

An identical 2D FFT (Two Dimensions FFT) image reconstruction was made with Connex, too. As we had to reconstruct a 128x128 image, we arranged horizontally our FID data into 128 float vectors (X[0] … X[127]) into Connex Array, computing first 128 FFTs on each vector (Fig. 6).

X[0]	x(0)	x(1)	.	.	.	x(127)
X[1]	x(0)	x(1)	.	.	.	x(127)
X[2]	x(0)	x(1)	.	.	.	x(127)
.	.	.				.
.	.	.				.
.	.	.				.
X[127]	x(0)	x(1)	.	.	.	x(127)

Fig. 8. Vector data disposal in Connex Array

For the vertical FFT, data must be filled in horizontally again, so matrix in Fig. 8 was transposed using two special functions (write(X,128,buffer) to save data in a temporary buffer, and read (X,128,buffer,128,1) to effectively compute the transposed matrix).

X[0]	x(0)	.	x(127)	x(128)	.	x(255)	.	x(895)	.	x(1023)
X[1]	x(0)	.	x(127)	x(128)	.	x(255)	.	x(895)	.	x(1023)
X[2]	x(0)	.	x(127)	x(128)	.	x(255)	.	x(895)	.	x(1023)
.	.	**Image 1**	.	.	**Image 2**	.		.	**Image 8**	.
.	.		.	.		.		.		.
.	.		.	.		.		.		.
X[127]	x(0)	.	x(127)	x(128)	.	x(255)	.	x(895)	.	x(1023)

Fig. 9. MRI data in Connex Array

Although a single image reconstruction (128x128) was made, our CA1024 chip [7] can compute 8 images in parallel (for *VECTOR_SIZE* = 1024), as can be seen in Fig. 9. Briefly, presented FFT CUDA algorithm is not as effective as 2D FFT algorithm implemented in VectorC with Connex Array.

Table 4

2D DFT (128x128 pixels) image reconstruction

Function	Floating point operations	Number of cycles	Execution time Connex Array
1D FFT,N=128 Im{X}=0	32943	35	87.5 ns
2D 128x128 Image Reconstruction Im{X}≠0,Re{X}≠0	168668816	573440	0.67 ms

Table 5 shows usual MRI computational performances compared to Connex Array.

Table 5

MRI scanner comparison

MRI SCANNER	PERFORMANCE
Siemens Magneton Simphony	35 frame/s, 128x128 pixels
GE Signa Ovation	150 frame/s, 128x128 pixels
Connex Array	350 frame/s, 128x128 pixels

5. Conclusions

FFT has been implemented in GPUs before. We tooked from [1, 17, 21, 22, 23] all data necessary for *Table 6* comparisons, reviewing the Fourier Transform and the classic FFT Cooley-Tukey algorithm used in MRI (and ultrasonic, X-Ray) image reconstruction on several GPUs, showing that for the same algorithm implemented on other parallel technologies, Connex's computing performance and power consumed are remarkable. Regarding the task efficiency (*Table 6*), one can see that Connex is eight times better than NVIDIA due to vectorial parallel implementation (shown in chapter 4), even though NVIDIA has a better general performance. As for TMS320C4X, computing time / task efficiency is obviously better since it's a specialized digital signal processor and Connex a general purpose parallel machine. Same remark for Cyclops-64 (specialized signal-processing chip with dedicated floating-point units, hence the power consumed is huge compared to Connex).

Table 6

FFT parallel implementation efficiency on different GPUs

Parallel computing machine	Algo-rithm	Core Speed (MHz)	Performance (Gflops)	Data size (pixels)	Computing time (ms)	Task efficiency (Gflop/task)	Power consumed (W)	Task energy efficiency (Watt/task)
NVIDIA Quadro FX NV 40	2D FFT	400	486	128 x 128 32-bit float	1.17	0.56	25	21.36
TMS320C4 DSP	2D FFT	400	147	128 x 128 32-bit float	0.42	0.061	5	11.9
Connex Arra	2D FFT	400	117	128 x 128 32-bit float	0.67	0.078	2.5	3.73
IBM Cyclops-64	2D FFT	500	20	128 x 128 32-bit float	3.47	0.069	83.22	23.98
BOBS 2040XL DS	2D FFT	150	1.3	128 x 128 32-bit float	60.3	0.078	2	0.033

In conclusion, we can say that MRI reconstruction and any other FFT-based applications are suitable for Connex Array's computational power and

energy efficiency. We leave this issue open and we remain receptive to any suggestion and information related to this paper.

Acknowledgement

The author got lot of technical and moral support from Gheorghe M. Ştefan, Peter Bluemler, Bogdan Mîţu, Radu Hobincu, Rustem Popa, Ana-Maria Calfa, Maria Aniţas, Flavia Ţugui.

R E F E R E N C E S

[1] *Long Chen, Ziang Hu, Junmin Lin, Guang R. Gao*, Optimizing the Fast Fourier Transform on a Multi-core Architecture, Computer Architecture and Parallel Systems Laboratory, University of Delaware, USA, 2007

[2] *D.M.S.L. Johnsson, R.L. Krawitz, R. Frye*, A radix 2 FFT on the connection machine. In Proceedings of Super computing, 1989, pag. 809-819

[3] *J.W Cooley, J.W. Tukey*, An Algorithm for the Machine Calculation of Complex Fourier Series. Math. Computation, 1965 , pag. 19, 297–301

[4] *E.Mark Haacke*, Understanding Magnetic Resonance imaging. Magnetic Resonance in medicine, Washington University, School of Medicine, Mallinckrodt Institute of Radiology, May 1999, Vol. 41, pag. 855-915

[5] *Ge Way, Michael W. Vannier*, Low contrast resolution in volumetric X-ray CT – Analytical comparison between conventional and spiral CT, University of Iowa, Department of Radiology, november 1996

[6] *Silvia De Francesco*, *Augusto Silva*, Fourier Methods in CT: projection and reconstruction algorithms, IEETA/DET Universitade de Aveiro, Campus Universitario, Portugal

[7] *Gheorghe Ştefan,* Integral Parallel Architecture In System-On-Chip Designs, Faculty of Electronics, Tc. and IT, Politehnica University of Bucharest, România

[8] *Adam Alessio, Paul Kinahan*, PET Image Reconstruction, University of Washington, Department of Radiology, USA

[9] *Jeroen Verhaeghe, Dimitry Van De Ville, Ildar Khalidov, Yves D'Asseler, Ignace Lemahieu, Michael Unser*, Dynamic PET Reconstruction Using Wavelet Regularization With Adapted Basis Functions, IEEE Transactions on Medical Imaging, Vol. 27, July 2008, pag. 948-958

[10] *Yao Xie*, Fourier-based forward and back projectors for iterative tomographic image reconstruction, Stanford University, December 2007

[11] *Yao Wang*, Computed Tomography, Polytechnic University Brooklyn, NY 11201

[12] *Matt A.Bernstein, Kevin F.King, Xiaohong Joe Zhou,* Handbook of MRI, 2004 , ISBN –13: 978-0-12-092861-3, pag. 5-15, 256-266, 378-380, 394-400, 405-410, 491-501

[13] *Dirk Alexander Sennst, Bernnard Schmidth, Oliver Watzke, Willi A. Kalender*, An extensible Software-based Platform for Reconstruction and Evaluation of CT Images, RadioGraphics Vol. 24, pag. 601-613, March 2004

[14] *J. L. Herraiz, S España, J. J. Vaquero, M. Desco, J M Udías*, FIRST: Fast Iterative Reconstruction Software for (PET) Tomography, Institute of Physics Publishing, Physics in Medicine and Biology, 2006, pag. 4547-4565

[15] *Philipp Kegel, Maraike Schellnann, Sergei Gorlatch*, Challenges and Approaches in Parallelizing Applications for Medical Imaging, University of Munster, Department of Computer Science, Germany, 2009

[16] *I.A. Cunninghan, P.F. Judy*, The Biomedical Engineering Handbook: Second Edition, Ed. Joseph D. Bronzino, Boca Raton: CRC Press LLC, 2000, pag. 385-443
[17] *Rose Marie Piedra*, Parallel 1-D FFT implementation With TMS320C4x DSPs, February 1994
[18] *Mihaela Maliţa, Gheorghe M. Ştefan*, Many-processors & KLEENE's model, UPB. SCi. Bull. Series C, Vol. 72, Iss. 3, 2010, ISSN 1454-234x
[19] *Bogdan Mîţu*, C Language Extension for Parallel Processing, BrightScale research report, 2008, http://arh.pub.ro/gstefan/VectorC.ppt
[20] *Istvan Lorentz*, Optimizing The Fast Fourier Transform on a many-core processor - the Connex Array, Transilvania University Braşov, March 2010
[21] *Nikos P. Pitsianis, Gerald Pechanek*, High-performance FFT implementation on the BOPS ManArray parallel DSP, SPIE Conference on Advanced Signal Processing Algorithms, Architectures and Implementations IX, Denver, Colorado USA, July 1999, SPIE Vol. 3807, pag. 166-170
[22] *Elkin Garcia, Ioannis E. Venetis, Rishi Khan, Guang R. Gao*, Optimized Dense Matrix Multiplication on a many-core Architecture, EURO Par 6272 LNCS, September 2010, pag. 316-327
[23] *Thilaka Sumanaweera, Donald Liu,* GPU Gems 2 programming techniques for high-perfoance graphics and general-purpose computation, Siemens Medical Solutions USA, 2005, ISBN 0-321-33559-7
[24] http://www.eclipse.org/downloads/