PENGEMBANGAN ALGORITMA UNTUK PERCEPATAN PROSES KOMPUTASI PARALEL PADA METODE *STAGGERED GRID FINITE DIFFERENCE TIME DOMAIN* MENGGUNAKAN ARSITEKTUR CUDA GPU

Arief Budiman [1)] , Joko Triono [2)]
[1). 2)] Dosen Fakultas Teknik Universitas Merdeka Madiun
email : arief@unmer-madiun.ac.id,, jokotriono2002@gmail.com,

Abstract

The utilization of parallel computing architecture with cuda gpu has been shown to speed up the computing process time, hence many scientific activities, database application, geometry application, image processing application use parallel architecture to get better computing performances.However computing time speed was also influenced by the proper algorithms, so then in this research will be developed an algorithm program that would be expected to gain more computing speed time process. On this research will be used a method of 2 dimention stagerred grid finite difference time domain.And tools used in this research is c cuda programming language with microsoft visual studio 2007.

Kata kunci : GPU, CUDA, 2D *Finite Difference Time Domain Stagerred Grid*

Latar Belakang

Salah satu pendekatan numerik yang dapat digunakan untuk memodelkan beberapa fenomena fisis yaitu dengan menggunakan metode numerik seperti *Finite Difference Time Domain* (FDTD) dan *Digital Waveguide Mesh*.

Akan tetapi pemodelan dengan menggunakan FDTD tersebut memiliki kebutuhan komputasi yang cukup berat, sehingga untuk menjalankan pemodelan tersebut akan diperlukan waktu komputasi yang cukup lama dan kebutuhan spesifikasi perangkat keras yang tinggi. Permasalahan tersebut sebenarnya dapat diselesaikan dengan menggunakan konsep komputasi paralel, dimana proses komputasi dapat dibagi ke beberapa komputer yang dirangkai dan terhubung satu sama lain dalam sebuah jaringan. Akan tetapi dengan menyediakan beberapa komputer yang kemudian saling terkoneksi dalam sebuah jaringan akanmembutuhkan biaya yang cukup tinggi.

Dalam beberapa tahun terakhir perkembangan arsitektur GPU berkembang cukup pesat, dan GPU pada masa kini dapat diprogram untuk melakukan proses komputasi, bahkan memiliki kemampuan komputasi yang lebih baik dari CPU sehingga hal ini dapat menjadi solusi untuk mengatasi masalah komputasi yang cukup berat pada penggunaan metode FDTD. Seperti yang telah ditunjukan oleh Kirk dan Hwu(2010) yaitu, konsep untuk memanfaatkan GPU seperti komputer paralel, dimana dengan arsitektur GPU masa kini memungkinkan untuk memiliki banyak *core* dalam sebuah prosesor GPU

Arsitektur komputasi paralel pada *Graphic Processing Unit* (GPU) yang saat ini banyak digunakan adalah *Compute Unified Device Architecture* (CUDA).CUDA memiliki 3 komponen

dasar untuk mengatur proses pengolahan data paralel, komponen tersebut adalah CUDA *driver*, CUDA *Application Programming Interface* (API), dan CUDA *mathematical libraries*. Ketiga komponen tersebut berjalan menggunakan *compiler* C (OČKAY dkk, 2008). Lebih dari sebelas tahun pemanfaatan GPU untuk proses komputasi telah diterapkan pada berbagai kegiatan ilmiah, seperti aplikasi *database*, geometri, aplikasi pengolahan citra, dan kesemuanya diterapkan dalam rangka untuk mendapatkan performa komputasi yang lebih baik (Savioja dkk, 2010).

Akan tetapi percepatan proses komputasi pada GPU selain dipengaruhi oleh spesifikasi dan kemampuan *hardware* GPU juga dipengaruhi oleh faktor penyusunan algoritma program. Seperti yang ditunjukkan oleh OČKAY et al (2010), bahwa untuk mendapatkan performa yang baik itu tidak hanya sekedar dapat menerapkan suatu kasus pada GPU tetapi juga harus memperhatikan batasan kemampuan *hardware* dan alokasi memori yang mana hal tersebut ditentukan pada penentuan algoritma kernel. Oleh karena itu perlu dirumuskan algoritma yang tepat untuk dapat lebih mempercepat proses komputasi paralel pada metode 2D *Staggered Grid* FDTD dengan menggunakan arsitektur CUDA GPU.

Tinjauan Pustaka

Staggered Grid Finite Difference Time Domain

Metode FDTD yang digunakan pada penelitian ini menggunakan metode Yee (Yee, 1966) yang pernah digunakan untuk menyelesaikan persamaan Maxwell untuk perambatan gelombang elektromagnetik menggunakan skema leapfrog pada S*taggered Cartesian Grids*. Metode ini dapat digunakan pada kasus akustik dengan menggunakan orde pertama dimana kecepatan (*Velocity*) dan tekanan (*Pressure*) ditambah dengan persamaan differensial.Dimana *u* dan *v* adalah *x* dan *y* dari komponen kecepatan partikel, dan *p* adalah *pressure*, ρ adalah densitas, *c* kecepatan suara. Dari persamaan tersebut kemudian diturunkan menjadi persamaan berikut :

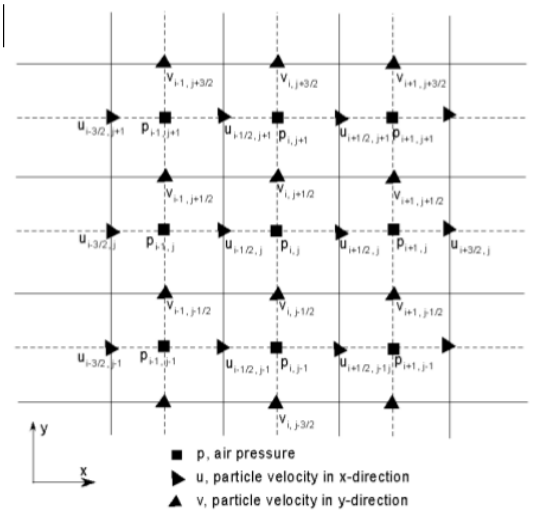
$$p_{i,j}^{n+1} = p_{i,j}^{n} - dt\left(\frac{u_{i+1,j} - u_{i,j}}{dx} + \frac{v_{i,j+1} - v_{i,j}}{dy}\right)$$

$$u_{i+1/2,j}^{n+1/2} = u_{i+1/2,j}^{n-1/2} - dt\left(\frac{p_{i+1/2,j}^{n} - p_{i,j}^{n}}{dx}\right)$$

$$v_{i,j+1/2}^{n+1/2} = u_{i,j+1/2}^{n-1/2} - dt\left(\frac{p_{i,j+1/2}^{n} - p_{i,j}^{n}}{dy}\right)$$

Dari Persamaan tersebut kemudian digunakan untuk mengembangkan kode FDTD 2 dimensi, langkah yang pertama pada kode FDTD 2 dimensi tersebut adalah menghitung jumlah nilai-nilai awal pada setiap langkah waktu, kemudian nilai awal tersebut dilakukan *looping* terhadap setiap satuan waktu selama proses komputasi. Dan dalam *looping* ini, yang dihitung adalah partikel kecepatan, tekanan, dan batas kondisi nilai.Pada gambar 1 ditunjukkan bagaimana persamaan diatas digunakan untuk menghitung nilai-nilai

dari *pressure* dan *velocity* dari setiap partikel dari nilai-nilai pada titik sebelumnya.

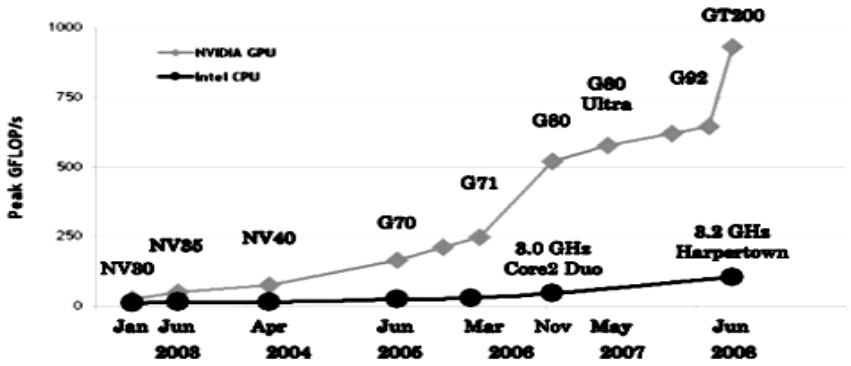


Gambar 1**.**Skema FDTD staggered Grid 2 dimensi pada sumbu x, y (Liu dan Albert, 2006).

Pada penelitian sebelumnya, *staggered grids* umumnya lebih banyak digunakan dibandingkan *unstaggered grids* untuk beberapa alasan. Seperti yang diungkapkan pada penelitian Gilles (2000), pada *unstaggered grids*, vektor medan elektrik dan magnetik terletak pada sumbu utama grid dan vektor medan dengan komponen terkolokasi. Pembedaan titik tengah pada *unstaggered grids* dapat menyebabkan osilasi numerik yang aneh pada proses *decoupling* (Press, 1992). Terlebih lagi pada skema *unstaggered grids* menghasilkan kesalahan empat kali lebih besar untuk fase kecepatan numerik bila dibandingkan dengan skema *staggered grids*. Dengan mengesampingkan masalah-masalah tersebut, *unstaggered grids* memiliki beberapa keuntungan bila digunakan untuk pemodelan fenomena gelombang optik. Sebagai contoh, skema *unstaggered grids* memungkinkan untuk menggunakan frame jendela koordinat untuk pelacakan denyut pada perambatan optikal pada jarak yang cukup jauh (Fidel dkk, 1997). Yang terpenting, penggunaan grid dengan komponen vektor medan terkolokasi efisien dan akurat untuk pemodelan perambatan gelombang optik non linier baik pada skema 2 dimensi ataupun 3 dimensi. Alasan dari fakta tersebut adalah bahwa pada pemodelan non linier membutuhkan perhitungan *high order* dari intensitas medan elektrik pada titik node di setiap satuan waktu. Jika komponen vektor medan elektrik tidak terkolokasi, seperti halnya dengan skema yee staggered, intensitas medan pada titik yang diberikan harus di interpolasi dari komponen medan yang berada di sekitarnya. Interpolasi ini cukup berat bila dilihat dari sisi efisiensi komputasional dan akurasi, dan dapat dihindari bila vektor medan elektrik terkolokasi.

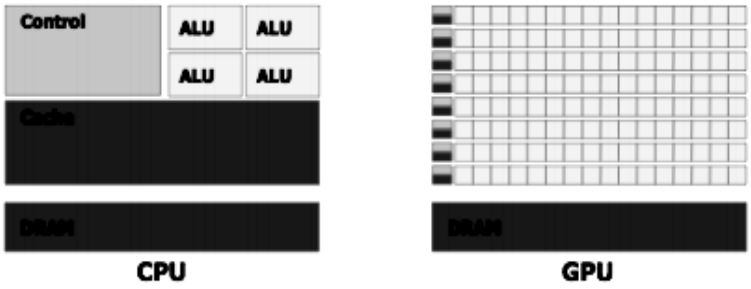
Compute Unified Device Achitecture (CUDA)

Ketika arsitektur prosesor *multicore* seperti quad coreIntel i7 berusaha untuk memaksimalkan kecepatan eksekusi program sekuensial, arsitektur core pada GPU juga mencoba untuk memaksimalkan kinerjanya untuk program komputasi paralel Kirk dan Hwu (2010).Kartu seperti Nvidia Tesla C1060 dan GT200 berisi 240 core, dan masing-masing bersifat multithreaded. Hal ini memungkinkan untuk kinerja yang jauh lebih besar dibandingkan dengan CPU tradisional, seperti dapat dilihat di sini:



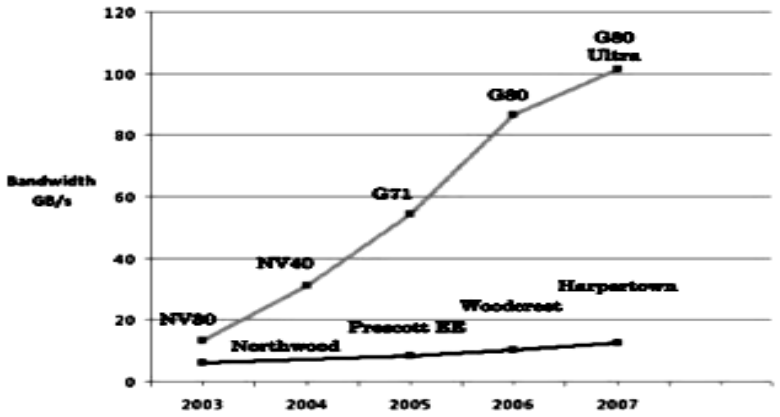
Gambar 2.Perbandingan kecepatan GPU dengan CPU (Webb, 2010).

Peningkatan kinerja ini disebabkan oleh desain hardware GPU yang berbeda dengan CPU.Dimana CPU multicore menyediakan cache besar dan melaksanakan instruksi x86 secara penuh pada setiap core, sedangkan GPU core yang lebih kecil yang ditujukan untuk throughput floating-point.



Gambar 3.Arsitektur CPU dan GPU (Webb, 2010).

Kinerja juga meningkat karena dukungan memory bandwithyang cukup besar.Kartu grafis memiliki 10 kali lebih besar dibandingkan dengan CPU. Dengan model terkini GPU dapat meiliki kecepatan hingga 100Gb/s.



Gambar 4.Perbandingan Memory Bandwith CPU dan GPU (Webb, 2010).

Pada tahun 2007 Nvidia meluncurkan teknologi CUDA untuk pertama kalinya.Bersamaan dengan hal itu pada tahun 2008 Apple's meluncurkan OpenCL, hal ini memungkinkan bagi para ilmuan untuk memanfaatkan potensi dari Komputasi GPU.OpenCL digunakan pada pada sistem operasi terbaru Apple, seperti Snow Leopard, dan menggunakan pemrograman berbasis C, C++, Apple Inc (2009).CUDA memerlukan SDK sebagai ekstensi untuk pemrograman menggunakan bahasa pemrograman C. Ini merupakan keuntungan untuk dunia pemrograman yang ingin mengembangkan komputasi GPU.

Hardware yang memiliki kemampuan dan fitur CUDA GPU memiliki serangkaian *streaming multiprocessor* yang memiliki kemampuan proses yang cukup tinggi, dan setiap *streaming multiprocessor* memiliki *streaming processor* yang membagi *control logic* dan *instruction cache*. Pada Tesla C1060 terdapat 30 *streaming multiprocessor* dan masing-masing memiliki 8 *streaming processor*, dimana setiap *streaming processor* dapat melakukan proses dalam jumlah yang banyak, dan dapat melakukan ribuan proses dalam setiap aplikasi.

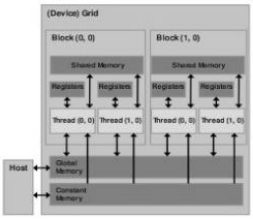
Ada beberapa tipe memori, setiap *core* memiliki *registers* dan *shared Memory* yang terdapat dalam satu chip dan memiliki akses yang sangat cepat. Dan ada juga *constant memory* yang juga cepat tetapi hanya *read only*, berguna untuk menyimpan parameter konstan. Dan yang terakhir adalah *Global Memory*, memiliki speed yang lambat tetapi ukurannya sangat besar. Bahkan Tesla C1060 memiliki *global memory* sebesar 4 gigabyte, *constant memory* sebesar 65 kilobyte, *register* dan *shared memory* sebesar 16 kilobyte pada setiap *streaming multiprocessor*.

Device code can :

- R/W per-thread registers
- R/W per-thread local memory
- R/W per-block shared memory
- R/W per-grid global memory
- Read only per-grid constant memory

Host code can :

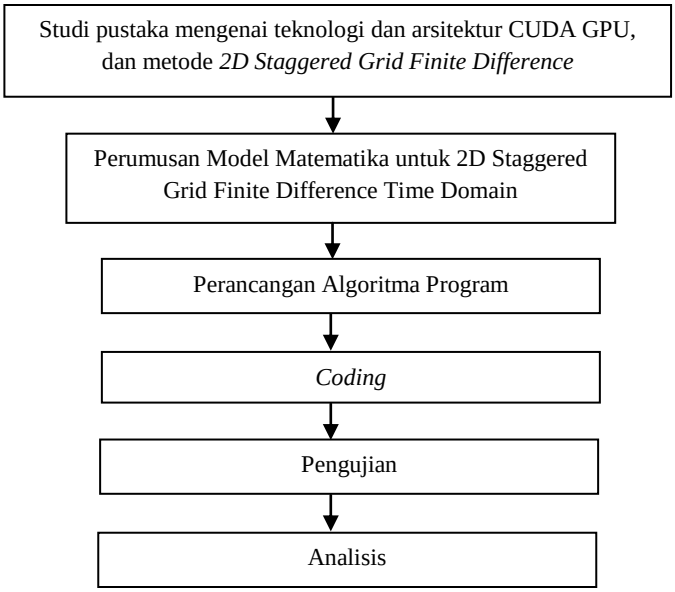
- R/W per-grid global and constant memory



Gambar 5. Arsitektur CUDA GPU (Webb, 2010).

Metodologi

Metode penelitian yang digunakan pada penelitian ini dapat dilihat pada *Flowchart* berikut :



Gambar 6. Flowchart Alur Penelitian

Langkah awal pada penelitian ini adalah dengan melakukan studi pustaka mengenai teknologi dan arsitektur CUDA GPU, begitu juga dengan pemodelan numeric menggunakan 2D *Staggered Grid Finite Difference Time Domain*.Dari dasar tersebut kemudian dirumuskan model matematika yang tepat untuk dapat diimplementasikan kedalam sebuah algoritma program sehingga dapat ditentukan algoritma yang kemudian diwijudkan menjadi sebuah program.

Setelah program telah berhasil dibangun kemudian dilakukan proses pengujian untuk masing-masing code program yang telah dibangun, untuk selanjutnya dianalisis sejauhmana hasil percepatan yang diperoleh dari masing-masing model program yang dibangun.

Implementasi

Langkah pertama yang dilakukan adalah, menginisialisasi *block threads* yang akan digunakan pada proses. Kedua, menginisialisasi nilai *p*, *u*, dan *v* pada memori *host*. Ketiga, mentransfer data *p*, *u*, dan *v* dari memori *host* ke memori GPU. Keempat, dengan data yang telah di *copy* memori GPU kemudian kernel dijalankan secara iteratif pada nilai *n* waktu. Kelima, setelah seluruh *threads* selesai data kemudian ditransfer kembali dari memori GPU ke memori *host*. Dan terakhir, mengosongkan memori pada memori GPU.

```
int x = threadIdx.x + blockIdx.x * blockDim.x;
int y = threadIdx.y + blockIdx.y * blockDim.y;
int offset = x + y * blockDim.x * gridDim.x;
```

Algoritma 1. Inisialisasi *block threads*.

Kemudian langkah selanjutnya adalah menginisialisasi nilai pada setiap titik grid sebagai kernel utama, dimana nilai *p*, *u*, dan *v* di inisialisasi dan dilakukan looping pada setiap satuan waktu dan nilai update pada setiap satuan waktu akan digunakan pada proses berikutnya. Nilai *u* dan *v*akan dilakukan masing-masing pada setiap iterasi.

```
__global__ void pressure_kernel(float *p,float *u,float *v)
{
// pressure calculation (p)
// map from threadIdx/BlockIdx to pixel position
int x = threadIdx.x + blockIdx.x * blockDim.x;
int y = threadIdx.y + blockIdx.y * blockDim.y;
int offset = x + y * blockDim.x * gridDim.x;
int right = offset + 1;
int top = offset + DIM;
p[offset] = p[offset] - dt*((u[right]-u[offset])/dx+ (v[top]-v[offset])/dy);
}
__global__ void u_vel_kernel(float *p,float *u)
{
// Horizontal (x-coord) particle velocity calculation (u)
// map from threadIdx/BlockIdx to pixel position
int x = threadIdx.x + blockIdx.x * blockDim.x;
int y = threadIdx.y + blockIdx.y * blockDim.y;
int offset = x + y * blockDim.x * gridDim.x;
int left = offset - 1;

if ((x > 0) && (x < DIM))
{u[offset] = u[offset] - dt*(p[offset]-p[left])/dx; }
}
__global__ void v_vel_kernel(float *p,float *v)
{
```

```
// vertical (y-coord) particle velocity calculation (v)
// map from threadIdx/BlockIdx to pixel position
int x = threadIdx.x + blockIdx.x * blockDim.x;
int y = threadIdx.y + blockIdx.y * blockDim.y;
int offset = x + y * blockDim.x * gridDim.x;
int bottom = offset - DIM;
if ((y > 0) && (y < DIM))
{ v[offset] = v[offset] - dt*(p[offset]-p[bottom])/dy;}}
```

Algoritma 2.Inisialisasi *kernel*.

Kemudian hasil nilai update setiap titik di copy dari CPU ke *block threads* pada GPU, untuk model program dengan menggunakan kernel yang pertama maka *code* untuk meng*copy* data ditulis sebagai berikut (algoritma 3) :

```
v_vel_kernel<<<blocksuv,threads>>>(d->p,d->u,d->v );
v_vel_kernel<<<blocksv,threads>>>(d->p,d->u,d->v );
pressure_kernel<<<blocks,threads>>>(d->p,d->u,d->v );
```

Algoritma 3. Alokasi *threads* pada masing-masing *block*.

Hasil dari komputasi menggunakan algoritma berbasis CPU dapat dilihat pada tabel 1 berikut :

Tabel 1. Hasil Pengujian Program Berbasis CPU

DIM (Ukuran Dimensi)	Waktu
800 x 800	159,75 s
1200 x 1200	388,11 s
1600 x 1600	677,52 s
2000 x 2000	967,14 s
2400 x 2400	1509,50 s
2800 x 2800	1540,65 s
3200 x 3200	1885,54 s
3600 x 3600	2242,91 s

Pada tahap pelaksanaan penelitian saat ini telah dirumuskan algoritma program berbasis CUDA dengan menggunakan Global Memory dan Texture Memory. Pada algoritma program berbasis CUDA inisialisai nilai *p,u,* dan *v* pada masing-masing *block thread*, tidak lagi direpresentasikan dalam bentuk array seperti pada algoritma program berbasis CPU. Untuk hasil perbandingan waktu komputasi antara algoritma program berbasis CPU dan GPU dapat dilihat pada Tabel 2 dan percepatan terhadap *code* program berbasis CPU dapat dilihat pada tabel 5.

Tabel 2. Hasil Pengujian Program Berbasis CUDA GPU

DIM (Ukuran Dimensi)	CUDA (Global Memory)	CUDA (Texture Memory)	CUDA (Graphic Interopability)
800 x800	18,29 s	16,52 s	16,71 s
1200 x 1200	39,39 s	36,61 s	36,88 s
1600 x 1600	72,48 s	65,96 s	66,71 s
2000 x 2000	108,44 s	101,21 s	101,96 s
2400 x 2400	163,12 s	147,61 s	148,96 s
2800 x 2800	213,5 s	198,94 s	201,00 s
3200 x 3200	287,46 s	262,11 s	264,59 s
3600 x 3600	357,18 s	333,56 s	337,38 s

Tabel 3.Perbandingan Percepatan Pada CUDA GPU dengan CPU

DIM (Ukuran Dimensi)	CUDA (Global Memory)	CUDA (Texture Memory)	CUDA (Graphic Interopability)
800 x800	8,73	9,67	9,56
1200 x 1200	9,85	10,60	10,52
1600 x 1600	9,34	10,27	10,15
2000 x 2000	8,91	9,55	9,48
2400 x 2400	9,25	10,22	10,13
2800 x 2800	7,21	7,74	7,66
3200 x 3200	6,55	7,19	7,12
3600 x 3600	6,27	6,72	6,64

Kesimpulan

Berdasarkan pengembangan algoritma program yang telah dilaksanakan sejauh ini, dapat ditarik kesimpulan sebagai berikut :

1. Dari hasil pengujian waktu komputasi antara algoritma program berbasis CPU dan CUDA GPU terlihat perbandingan percepatan waktu yang cukup besar, mencapai angka 10 kali lebih cepat.
2. Penggunaan model *Global Memory* dan *Texture Memory* juga mempengaruhi percepatan waktu komputasi, dimana pada *Texture Memory* terlihat lebih cepat bila dibandingkan dengan menggunakan *Global Memory*.

Daftar Pustaka

Apple Inc., 2009, OpenCL programming guide for OSX, (http://developer.apple.com/mac /library/documentation/Performance/ Conceptual/OpenCL_MacProgGuide/ Introduction/Introduction.html.

Fidel, B., Heyman, E., Kastner, R., Ziolkowski, R. W., 1997, Hybrid ray-FDTD moving window approach to pulse propagation, J. Comput.Phys.138, 480.

Gilles, L., Hagness, S. C., V´azquezy, L., 2000, *Comparison between Staggered and Unstaggered Finite-Difference Time-Domain Grids for Few-Cycle Temporal Optical Soliton Propagation*, Journal of Computational Physics161,379–400.

Kirk, D., Hwu, W., 2010, *Programming*

Massively Parallel Processors, Morgan Kaufmann Publishers.

Liu, L., Albert, D.G., 2006, Acoustic pulse propagation near a right angle wall. J.Acoust. Soc. Am., 119, 2073-2083.

Press, W.H., 1992, Numerical Recipes in Fortran: The Art of Scientific Computing, 2nd ed. (Cambridge, Univ. Press, England).

OČKAY, M., HARAKAĽ, M., LÍŠKA, M., 2008, *Compute Unified Device Architecture (CUDA) GPU Programming Model and Possible Integration to the Parallel Environment*, Science & Military Journal.

OČKAY, M., HARAKAĽ, M., LÍŠKA, M., 2010, *Kernel Dependencies in a Modern General-Purpose GPU Architecture*, Science & Military Journal.

Savioja, L., Manocha, D., Lin, M.C., 2010, *Use of GPUs in room acoustic modeling and auralization*, Proceedings of the International Symposium on Room Acoustics, ISRA.

Webb, C.J., 2010, *Computing 3D Finite Difference schemes for acoustics – a CUDA approach,* MSc Thesis University of Edinburgh College of Humanities and Social Sciences School of Arts, Culture and Environment.

Yee, K. S., 1966, Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media. IEEE Trans. Antennas Propag.,14, 302–307.