MODERN HARDWARE TO SIMULATE TSUNAMI WAVE PROPAGATION

Mikhail M. Lavrentiev
Sobolev Institute of Mathematics, Novosibirsk, Russia,
mmlavr@nsu.ru

Alexey A. Romanenko
Novosibirsk State University, Novosibirsk, Russia,
arom@ccfit.nsu.ru

ABSTRACT

Problem of tsunami risks evaluation, assessment and mitigation is here addressed. Modern computational technologies are able to calculate accurately tsunami wave propagation over the deep ocean provided that initial displacement (perturbation of the sea bed at tsunami source) is known. Modern deep ocean tsunameters provide direct measurement of the passing tsunami wave in the real time mode, which help to estimate initial displacement parameters right after the tsunami wave is recorded at one of the deep ocean buoys. Therefore, fast tsunami propagation code that can calculate tsunami evolution from estimated model source becomes critical for timely evacuation decision for many coastal communities in case of a strong tsunami.

In this paper we discuss a part of MOST (Method of Splitting Tsunami) software package, which has been accepted by the USA National Ocean and Atmosphere Administration as the basic tool to calculate tsunami wave propagation and to evaluate of inundation parameters. Our main objectives are to speed up the existing sequential program, and to adapt this program for shared memory systems (OpenMP), CELL BE architecture and GPU. For caring out this research we use SMP server and a system build on IBM CELL BE CPU. We also use NVIDIA Tesla™ C1060 board which acts as co-processor for CPU. Optimization of the existing parallel and sequential code for the task of tsunami wave propagation modeling as well as an adaptation of this code for GPU and CELL BE processor is discussed. The obtained results are up to 170 times performance gain for one time step. These are very promising.

KEYWORDS

Tsunami simulation, MOST, Splitting Tsunami, optimization, paralleling, HPC.

1. INTRODUCTION

Throughout the whole history of mankind people living on an ocean shore have from time to time undergone the attacks of catastrophic sea waves – tsunami, that are most often the result of powerful submarine earthquakes and landslides. The mankind is unable to countervail this managing natural disaster yet. Various protective structures represented by mighty concrete walls were actually put up in Japan, the state most vulnerable to tsunami attacks. These structures can lessen the damage inflicted by tsunami waves of a small height, but are hardly of any use in case of catastrophic tsunami the height of which can gain an altitude of 35 meters. That is why nowadays the most efficient measures are evacuation efforts, which should be underway in case of a real tsunami threat that must be reported by the tsunami warning service (local or national). In this regard, the sooner the alert, the better. Taking into account the fact that tsunami waves appear as a result of powerful submarine earthquakes, tsunami watch can issue alert as soon as seismometers detect any fairly strong submarine earthquake. Nevertheless, such an approach leads to too many false alarms with evacuation expenses by an order(s) exceeding material losses incurred by the wave attack. Several false alarms may cause the population to develop alarm immunity which will result in casualties after a truly dangerous tsunami wave. Moreover, high tsunami waves may originate from a submarine landslide brought about by a relatively weak earthquake. Therefore, tsunami warning services should not ignore weak earthquakes as well.

At the turn of the 20th and the 21st centuries it finally became technically possible to detect tsunami waves in the mid-ocean and to transmit these data via satellite-delivered channels in real-time mode. In addition, recording devices transmitting data concerning the deviation of water surface from zero level can be located practically anywhere in the ocean. This allows to register and to measure the actually emerged tsunami waves soon after they have formed as a result of the initial displacement of water surface in tsunami source. Deep water (mid-ocean) measurements are free from distortions associated with friction on the ocean floor and multiple reflections from the shoreline. Thus, these data contain more information about the tsunami source, and their processing allows to improve the reliability of this tsunami hazard chance prognosis. The developed mathematical methods allow approximate definition of source parameters according to the data from several sub-oceanic recording stations in real-time mode, that is in 1-2 minutes after the detection of the wave by two or more sub-oceanic stations.

Numerical computations of tsunami wave propagation process from the source to the shore are now widely used for the purposes of estimating the potential height of tsunami waves and defining inundation zones. Tsunami source configuration and the scale of the initial vertical displacement of the water surface in it essentially influences the characteristics of the wave approaching the shore. That is why, while trying to solve the direct problem of tsunami propagation (from the known source to the points on the coast), it is necessary to on-line estimate several case scenarios with different source parameters as soon as possible (favorable within several minutes). The fact that it is necessary to carry out several calculations is incurred by the ambiguousness of tsunami source parameters definition using the data of a small number of recording stations. The data regarding the source should be improved with the origination of new data as a tsunami propagates through a greater number of sub-oceanic recording devices. Therefore, the performance of numerical calculations is of crucial importance. The less time is spend on estimating several scenarios of tsunami propagation in the considered area of responsibility, the more time is left to undertake a decision and to carry out evacuation efforts in the coast subsections if required due to the estimated wave height.

Efforts to perfect on-line tsunami prognosis using sub-ocean recording devices became much more intense after the catastrophic tsunami in the Indian Ocean on December 26, 2004 that led to the death of 250000 people. As a matter of fact, it would have been possible to avoid (or at least to reduce the number of) multi-thousand victims on the coasts of India and Sri Lanka if at least one sub-oceanic recording station had been located the Bay of Bengal by that time. Then there would have been fairly much time to work out a realistic prognosis of the estimated tsunami heights as well as potential inundation zones of the shore and to issue tsunami alarm and evacuation of the population to secure locations (high elevation terrains or reasonably durable constructions of a necessary height). It is worthy of note that several sub-oceanic stations transmitting data regarding the ocean level state in real-time mode have been located and are functioning in that region of the Indian ocean by now.

2. MATHEMATICAL STATEMENT OF THE PROBLEM

The approximations of shallow-water theory (both lineal and non-lineal) are used as the basic models for describing wave propagation throughout the ocean. These models rather accurately reflect basic wave parameters (propagation time period from the source to the recording device and wave amplitudes) even for a fairly rough numerical bathymetry in the assumption that initial displacement in the source is unknown. There are several software packages for modeling wave propagation throughout the ocean and the run-up heights. The most well-known packages are MOST and TUNAMI.

MOST (Method of Splitting Tsunami) [1,2] allows to make a forecast of the flood region in real-time mode using tsunameters' data. This package is mostly used in USA for charting inundation maps [3]. Online version of MOST – comMIT – has also been created. TUNAMI N2 package for the Tsunami Inundation Modeling Exchange (TIME) program was worked out by Imamura in 1993. The registered copyright holders for this package are professors Imamura, Yalziner, and Synolakis. TUNAMI N2 has been successfully used for analyzing some tsunami [4,5]. All speculations hereafter will refer to the MOST package only.

The MOST software package uses numerical model of calculating wave propagation through deep water zone applying decomposition method for spatial variables. This method was initially developed in the Tsunami Laboratory, Computer Center of the Siberian Division, Academy of Sciences of USSR in Novosibirsk. Then the method was updated in the Pacific Marine Environmental Laboratory (NOAA, Seattle, USA) and was adapted to the models and standards of data accepted by tsunami watch services in the United States as well as other countries and used in tsunami research works in most countries. MOST is used to numerically simulate three processes of tsunami evolution: the estimation of residual displacement area resulting from an earthquake and tsunami production, transoceanic propagation through deep water zones, and contact with land (run-up and inundation). The given research work is concerned with the second stage – deep water wave propagation.

Nonlinear approximation of shallow water system is used for numerical calculation of tsunami wave propagation as follows [6]:

$$H_t + (uH)_x + (vH)_y = 0,$$

$$u_t + uu_x + vu_y + gH_x = gD_x, \qquad (1.1)$$

$$v_t + uv_x + vv_y + gH_y = gD_y,$$

Where H(x, y, t) = h(x, y, t) + D(x, y, t), h - stands for the height of the wave calculated from unperturbed level, D – the function delineating bottom configuration (digital bathymetry), u(x, y, t), v(x, y, t) – speed vector components along x and y respectively, and g – acceleration of gravity.

The adduced shallow water model soundly describes the process of tsunami waves transoceanic propagation providing that the horizontal dimension of ocean floor surge by an order exceeds the ocean depth at that point.

System (1.1) could be presented in symmetric form:

$$\frac{\partial z}{\partial t} + A\frac{\partial z}{\partial x} + B\frac{\partial z}{\partial y} = F ,\qquad (1.2)$$

where

$$z=\begin{vmatrix} u \\ v \\ H \end{vmatrix}, A=\begin{vmatrix} u & 0 & g \\ 0 & u & 0 \\ H & 0 & u \end{vmatrix}, B=\begin{vmatrix} v & 0 & 0 \\ 0 & v & g \\ 0 & H & v \end{vmatrix}, F=\begin{vmatrix} gD_x \\ gD_y \\ 0 \end{vmatrix}$$

The rectangular zone $\Omega=\{x,y:0\le x\le X,0\le y\le Y\}$ with its sides parallel to coordinate axes will be viewed as the variation domain of spatial variables.

The numerical algorithm to solve the system (1,2) is based on spatial decomposition along axis directions. For this purpose, let us study two backup systems, each of which depends on one spatial variable only.

$$\frac{\partial \psi}{\partial tt} + A\frac{\phi}{\partial tx} = F_1, 0\le x\le X ;\qquad (1.3)$$

$$\frac{\partial \psi}{\partial tt} + A\frac{\psi}{\partial ty} = F_2, 0\le y\le Y ,\qquad (1.4)$$

where $F_1=\begin{vmatrix} gD_x \\ 0 \\ 0 \end{vmatrix}, F_2=\begin{vmatrix} 0 \\ gD_y \\ 0 \end{vmatrix}$

This approach has first been suggested in Tsunami Laboratory of Computing Center in Novosibirsk (now, Institute of Computational Mathematics and Mathematical Geophysics, Siberian Division of RAS).

In order to find numerical solution of the system (1.2), it will suffice to make numerically stable solution for systems (1.3) and (1.4). Let us make the difference scheme for the system (1.3). The equations for this system are put down as follows:

$$v_t + uv_x = 0,$$
$$u_t + uu_x + gH_x = gD_x,$$
$$H_t + (uH)_x = 0.$$

(1.5)

This is a quasi-linear hyperbolic system. All eigenvalues of matrix A are real and diverse:

$$\lambda_1=u, \lambda_{1,2}=u\pm\sqrt{gH}$$

We will use the canonical form of this system for the numerical solution. This allows to actualize boundary conditions for the finite-difference analogue of the boundary value problem. The canonical form is put down as follows:

$$v'_t + \lambda_1 = 0$$
$$p_t + \lambda_2 p_x = gD_x \qquad (1.6)$$
$$q_t + \lambda_3 q_x = gD_x$$

where v', p, q are Riemannian invariants of the system (1.5) which have the following form:

$$v' = v$$
$$p = u + 2\sqrt{gH} \qquad (1.7)$$
$$q = u - 2\sqrt{gH}$$

In order to find numerical solution of the system (1.6), it was suggested to make an explicit difference scheme on a four-point stencil with quadric

approximation order regarding spatial variables and first order approximation with respect to time.

Numerical calculation of tsunami wave propagation process beginning from its spatial source usually starts with stating initial conditions that represent water surface vertical displacement zone (having summed it up with depth we get the thickness of the water layer) and initial wave stream velocity components. As a rule, in case of submarine earthquakes this velocity is minor and can be neglected, that is, it may be considered that water is quiescent, though it is no longer in equilibrium condition because of some zone's vertical displacement. Then only boundary conditions are actualized for every calculation step. In practice, as a rule, only wave amplitude has a practical meaning. That is why outbound parameters of the algorithm are represented by the values of water surface elevation at all cross-points of computational grid at certain given instants and sequences of ocean level values at certain area points (calculation marigrams).

3. SOFTWARE MODEL

As indicated in the mathematical model noted above, the calculation of water propagation is performed in two stages: the first step consists in calculating water displacement along X-axis, the second step – in calculating water displacement along Y-axis. We employ directions splitting approach. The calculations for i^{th} and j^{th} line along the X-axis may be carried out independently. Calculations along y-axis are carried out likewise. The main calculation loop may be schematically represented as follows:

```
[for each time step]
  // do calculations along X-axes
  [for each column of research domain – i]
    [calculate invariants for column i]
    [process the invariants – function swater]
    [recalculate invariants back to physical
        values]
  // do calculations along Y-axes
  [for each row of research domain – i]
    [calculate invariants for row i]
    [process the invariants – function swater]
    [recalculate invariants back to physical
        values]

  [save results for current time step to disk]
```

Figure 1: The main calculation loop algorithm.

There are four matrices used for calculations: depth profile (d), wave height (q), wave propagation velocity (u), along X-axis and wave propagation velocity (v) along Y-axis. The dimension of matrix equals the dimension of calculation field and constitutes 2581x2879 for the Pacific Ocean aquatorium.

In the original sequential program for the purpose of optimization, the second part of time iteration requires matrices transposition for the functions for q, u and v. Matrix d (depth) is left unchanged and can be transposed once before the main loop.

The mail loop in C language is displayed in Figure 2.

```
for(n=0; n<mmax; n++){
    for(j = 0; j < n1; j ++) {  // calculate along Y
        for(i=0; i<n2; i++) {
            qw[i] = u[j*n2 + i] - 2.0f * sqrtf( 9.8f*q[j*n2 + i]);
            uw[i] = u[j*n2 + i] + 2.0f * sqrtf( 9.8f*q[j*n2 + i]);
            vw[i] = v[j*n2 + i];
        } //for i
        swater(uw, qw, vw, &d[j*n2], u1, q1, v1, &n2, h2, &grnd, &t);
        for(i=0; i<n2; i++){
            q[j*n2 + i] = (u1[i] - q1[i]) * (u1[i] - q1[i]) /(9.8f*16.0f);
            u[j*n2 + i] = (u1[i] + q1[i]) * .5f;
            v[j*n2 + i] = v1[i];
        } //for i
    } //  for j

    transpose(q, n2, n1); transpose(u, n2, n1); transpose(v, n2, n1);

    for(j=0; j<n2; j++){  // calculate along X
        for(i=0; i<n1; i++){
            qw[i] = v[j*n1 + i] - 2.0f * sqrtf( 9.8f*q[j*n1 + i]);
            uw[i] = v[j*n1 + i] + 2.0f * sqrtf( 9.8f*q[j*n1 + i]);
            vw[i] = u[j*n1 + i];
        } //for i
        swater(uw, qw, vw, &dt[j*n1], u1, q1, v1, &n1, h1, &grnd, &t);
        for(i=0; i<n1; i++){
            q[j*n1 + i] = (u1[i] - q1[i]) * (u1[i] - q1[i]) /(9.8f*16.0f);
            v[j*n1 + i] = (u1[i] + q1[i]) * .5;
            u[j*n1 + i] = v1[i];
        } //for i
    } //  for j

    transpose(q, n1, n2);   transpose(u, n1, n2);  transpose(v, n1, n2);
    // getting data from "sensors" and save result
}
```

Figure 2: Main computation loop for the sequential program in C language.

The number of main loop iterations can constitute about 9000. This corresponds to 24 real-time hours required for the wave propagation all over the ocean. All iterations contain approximately equal number of operations, therefore, execution time for one iteration may serve as acceleration indicator. So, execution time for one time iteration using Dual-Core AMD Opteron™ Processor 2218 MHz constitutes about 3,5 seconds. OpenMP program version for 4 cores executes one time iteration for about a second, the one for 8 cores – for 0,53 seconds. Cluster was also used to carry out calculations, however, the algorithm required frequent data exchange among cluster nodes, so the attempts to develop considerable acceleration failed – the maximum achieved was 4 times at 12 nodes.

The best results were achieved on the basis of SONY PlayStation3 (IBM CELL BE processor, 6 computation units used). With optimizations having been conducted, 50-fold acceleration compared to the original sequential program has been achieved.

Calculations along X-axis, and then along Y-axis can be performed independently for every row/column. Moreover, it is possible to compute the values of invariants and new values of wave height and velocity along the axes for each point of space independently. This gives ground to anticipate this algorithm to be effective at Graphics Processing Unit (GPU) architecture.

Using GPU for mathematical calculations became very popular recently. GPU can perform hundreds billions operations per second and can process up to 1000s threads simultaneously. Moreover, there are several approaches which allow one to program GPU easily (CUDA [7], OpenCL [8]). However, application of GPU is effective only under certain conditions.

Adaptation the algorithm to GPU

One of standard approaches to adapting programs to GPU consists in successive rewriting of code areas to GPU. The code on Fig 3 was suggested within the framework of the implementation of this approach.

For the sake of simplicity the selected block size (ThreadBlock) equals to 16x16 threads. Consequently, all modeling space is split into equal units. As subsequently established through profiling and modeling operations, this configuration is optimal for stream multiprocessors load. Furthermore, with the use of video card, execution time for one iteration constituted an average of 0.25 seconds. Optimization has not taken place. At the same time 43% of time was taken by SWater kernel, while transpose kernel took 26%.

```
for(int i=0; i<cfg.steps; i++){
    // calculate along X
    Invariants_X<<<dimGrid, dimBlock>>>(... , x_size, y_size);
    SWater_<<<dimGrid, dimBlock>>>(... , x_size, y_size);
    RInvariants_X<<<dimGrid, dimBlock>>>(... , x_size, y_size);
    // calculate along Y
    transpose<<<dimGrid, dimBlock>>>(d_qwdata, d_q1data, x_size, y_size);
    transpose<<<dimGrid, dimBlock>>>(d_uwdata, d_u1data, x_size, y_size);
    transpose<<<dimGrid, dimBlock>>>(d_vwdata, d_v1data, x_size, y_size);

    Invariants_Y<<<dimGrid_, dimBlock>>>(... , y_size, x_size);
    SWater_<<<dimGrid_, dimBlock>>>(..., y_size, x_size);
    RInvariants_Y<<<dimGrid_, dimBlock>>>(..., y_size, x_size);

    transpose<<<dimGrid_, dimBlock>>>(d_qwdata, d_q1data, y_size, x_size);
    transpose<<<dimGrid_, dimBlock>>>(d_uwdata, d_u1data, y_size, x_size);
    transpose<<<dimGrid_, dimBlock>>>(d_vwdata, d_v1data, y_size, x_size);
    // getting data from "sensors" and save result
    ...
}
```

Figure 3: Main computation loop, adapted to CUDA

Functions dealing with invariants and reversed invariants are trivial and do not require further comments. The implementation of transpose function was taken from CUDA SDK. SWater_ function turned out to be the most difficult in terms of its implementation. It presupposed many verifications of boundary conditions (such as the presence of continent and island shores) and execution of intensive data reading. In order to simplify the function, the computation run of some conditions was done in advance which allowed to eliminate embedded validation operations. Moreover, all data required for calculations were copied into shared memory to be worked with thereafter.

Modeling operations revealed some inaccuracy in invariant computation (kernels for Invariants_X and Invariants_Y) marked by kernels' containing **sqrt**() function call that brings up an error in 3 ulp, which eventually leads to spontaneous swaying of the ocean surface already with 50 iterations. Several solutions to be considered were: to calculate invariants in double precision, to represent double through 2 floats, or, alternatively, to eliminate squared root computation within the mail loop. The first two variants did not incur considerable failure in productive capacity. Each stream multiprocessor currently contains 8 modules for processing single-precision floats and 2 for double-precision floats. The final variant – the elimination of square root computation – turned out to be the simplest and the most suitable in terms of its implementation.

In order to avoid constantly having to execute square root computation, it was decided to make all calculations through invariants and to re-count invariants back to the values of wave height and direction velocities at modeling data retention stage. As a result, the code has been modified, the ocean surface stopped "swaying", and the execution time of one iteration constituted an average of 0.23 seconds per one time iteration. The program was accelerated by 10% with having replaced some division operators in SWater_ kernel with multiplication operators. According to documentation data, division takes nine times more clock cycles than multiplication: 36 vs. 4 clock cycles. The execution time of one iteration has therefore constituted 0.21 seconds.

Several ways of program optimization have been checked out:

1. to create two SWater_ functions that will conduct calculations along different directions and, therefore, to avoid transpose() function call.
2. to justify line beginnings of all matrices in GPU memory in order to get consistent read operation. Replace cudaMallock() function call with cudaMallockPitch().
3. switch to memory processing through textures.

As profiler show, transpose function takes about 26% of program execution time. At the same time, it is used just to create convenient data location in memory for SWater_ function at the second half of iteration. This is sound for the consecutive program version and the program at OpenMP, but is unnecessary for implementation at CUDA with account of further optimizations. This resulted in SWater_r function having been created. Executing the same calculations as SWater function along a different axis, it shortens one cycle execution time to 0.135 seconds. Moreover, the program itself has been simplified.

To define the ways of further optimization we use profiler. Analysis has shown that, notwithstanding the seeming simplicity of Inv2 and Inv3 functions, incoherent read and record operations take place 2.6 million times while those coherent – several thousand times only.

The variable of 2.6 millions incoherent read operations (gld_incoherent) is an average of the number of elements in qw, uw, and vw arrays from the example at Figure 5. So, all entries turn up leveled past the unit boundary. To rectify this, we will substitute memory allocation via cudaMallock() function for cudaMallockPitch(). As a result of this change, the quantity of incoherent memory calls has shortened to zero and the execution time of one iteration has constituted 0.037 seconds, which is 100 times better than the result for the coherent program and 14 times quicker than OpenMP program version used 8 cores. It is worth mentioning that the concurrent program version had row start leveling result in acceleration by merely portions of a percent.

```
Inv1<<<dimGrid, dimBlock>>>(... , x_size, y_size);
for(int i=0; i<cfg.steps; i++){
    // calculate along X
    SWater_<<<dimGrid, dimBlock>>>(..., x_size, y_size);
    Inv2<<<dimGrid, dimBlock>>>(..., x_size, y_size);
    // calculate along Y
    SWater_r<<<dimGrid, dimBlock>>>(..., x_size, y_size);
    // getting data from "sensors" and save result
    ...
    Inv3<<<dimGrid, dimBlock>>>(..., x_size, y_size);
}// for cfg.steps
```

Figure 4: Main computation loop after optimization

```
__global__ void Inv2(float *q, float *u, float *v,
                     float *qw, float *uw, float *vw, int width, int height){
    unsigned int xIndex = blockIdx.x * blockDim.x + threadIdx.x;
    unsigned int yIndex = blockIdx.y * blockDim.y + threadIdx.y;
    unsigned int indx = yIndex * width + xIndex;
    float bq, bu, bv;
    if((xIndex<width) && (yIndex<height)){
        bv = v[indx]; bq = q[indx]*0.5f; bu = u[indx]*0.5f;
        qw[indx] = bv - (bu - bq);
        uw[indx] = bv + (bu - bq);
        vw[indx] = bu + bq;
    }
}
```

Figure 5: Invariant recalculation kernel

Table 1: Quantity of coherent/incoherent read/write operations

method	gld_incoherent	gld_coherent	gst_incoherent	gst_coherent
SWater_	4.92214e+06	105505	5.24688e+06	48144
Inv2/Inv3	2.62344e+06	10935	5.24688e+06	43740
SWater_r	4.72e+06	91004	5.24688e+06	47772
RInv	2.62344e+06	10935	1.74896e+06	14580

Memory-related operations for CPU are effective in case memory data are located close to each other and can be easily placed in cache, or CPU can determine memory access pattern in advance. GPU presupposes textures to be used for locating data in cache. Cache can room data localized in two-dimensional space with regard to the addressed data.

Further optimization resolved itself to declaring 7 two-dimensional textures and 2 one-dimensional, textures binding to data storage area in GPU memory, and inconsiderable modification of all kernels. This optimization gave an advantage of an average of 0.03 seconds per iteration – about 10%.

For final test purposes, the program was run at NVIDIA Tesla C1060. The achieved result was 0.02 seconds per an iteration, so, the final acceleration was 170 times compared to the original sequential program . Notwithstanding the fact that Tesla C1060 has twice more kernels than GeForce 9800 GX2, double acceleration was not the result of 20% higher clock frequency of the video card.

4. CONCLUSION

In this paper we presented the optimization of tsunami propagation modeling program (module of the MOST package) and its transfer from the coherent version to GPU. One of two GPU of GeForce 9800 GX2 video card has been used for computations while NVIDIA Tesla C1060 card has been used for the final test.

Adaptation process consists of selecting the blocks that can run with maximum parallelism degree and rewriting of these blocks using CUDA technology. The procedure is as follows: we copy data to the device, start up the kernel that processes it, and return the result back to the host. That is how all blocks are transferred, which is followed by eliminating the function of host-to-system data transmittance. Such adaptation results in efficiency boost (slightly more than 10 times higher acceleration). Nevertheless, there is accuracy divergence in terms of approximate square root computation.

The next step consists in algorithm modification that allowed to operate on invariants and to avoid multiple necessity of square root computation. This has raised calculation accuracy and slightly accelerated the program. One of the functions has then been rewritten, which allows to avoid matrix transposition. That has also slightly accelerated the program.

Aligning array row beginnings (cudaMallockPitch usage) has resulted in the highest boost of efficiency in the course of optimization – four fold acceleration that originated from considerably contracted quantity of incoherent read and data record operations.

The total acceleration with the use of GeForce 9800 GX2 video card has become 100-fold compared to the result for the coherent program and twofold compared to the program on IBM CELL BE.

The total time required for modeling wave propagation through the Pacific Ocean aquatorium has been decreased from 8.5 hours to 3 minutes (i.e. by factor of 170) when employing personal computer facilities with an installed NVIDIA Tesla C1060 card.

REFERENCES

[1] Titov, V.V., 1989, Numerical modeling of tsunami propagation by using variable grid. *Proceedings of the IUGG/IOC International Tsunami Symposium*, 46-51. Computing center Siberian Division USSR Academy of Sciences, Novosibirsk, USSR.

[2] Titov V.V. and Synolakis, C.E., 1998 Numerical modeling of tidal wave runup, *Journal of Waterway, Port, Coastal and Ocean Engineering*, 124(4), 157-171. Borrero, J.C., Cho, S. Moore, J.E, Richardson, H.W., Synolakis, C.E., 2005 Could it happen here, *Civil Engineering*, 75(4), 55-67.

[3] Shuto, N., C. Goto, F. Imamura (1990), Numerical simulation as a means of warning for near field tsunamis, Coastal Engineering in Japan, 33(2), 173-193.

[4] Yalciner A. C. Alpar B., Altinok Y., Ozbay I., Imamura F., 2002, "Tsunamis in the Sea of Marmara: Historical Documents for the Past, Models for Future", Marine Geology, 2002, 190, pp:445-463

[5] Stoker, J. J., 1992, *Water Waves: The Mathematical Theory with Applications*, Wiley-interscience, United States.

[6] High-Precision Software Directory [http://crd.lbl.gov/~dhbailey/mpdist/]

[7] GPU Gems 3, Addison-Wesley Professional, 2007

[8] Maurice Herlihy, Nir Shavit, *The Art of Multiprocessor Programming (Paperback)*, Morgan Kaufmann, 2008