Journal of Theoretical and Applied Information Technology
31st December 2016. Vol.94. No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

SOLVING A PROBLEM OF RESOURCE-INTENSIVE MODELING OF DECODERS ON MASSIVELY PARALLEL COMPUTING DEVICES BASED ON VITERBI ALGORITHM

ALEXEY VIKTOROVICH BASHKIROV, ALEXANDER VASILIEVICH MURATOV, OLEG YURIEVICH MAKAROV, VASILY IVANOVICH BORISOV, KSENIA NIKOLAEVNA LAPSHINA

Voronezh State Technical University,
Moscow Avenue, 14, Voronezh, 394066, Voronezh region, Russia

ABSTRACT

In this paper, we consider the problem of resource-intensive simulation of coding/decoding which corrects errors made at the preliminary stages of modern telecommunication system development. We propose to use the technology of parallel computing on GPU (GPGPU) to solve the problem of the process acceleration. We discuss the aspects of encoding/decoding simulation, which corrects errors in heterogeneous systems. The results of this technology applying in the convolutional codec parameters simulation, decoded by Viterbi algorithm, are given as well. Another problem concerned with limitation of the interaction speed with the computing device tail part and a random access to memory is also considered. We propose a solution by communication minimization at host-computing device level, as well as the use of caching. The simulation tools are described in the paper, including the use of computing technique of general purpose on GPU allowing to reduce the time required to optimize the noiseless coding system and thus for the development and implementation of telecommunication devices. We describe the solutions of tasks on codecs characteristics research using massively parallel computing, differing by simplified initialization of flow pseudorandom-number generator (PRNG) ensuring high performance with sufficient accuracy of calculations by reducing the number of calls to an external status register.

Keywords: *Parallel Computing, Viterbi Algorithm, Noiseless Coding, GPU Of The Opencl Standard, Communication Channels, Heterogeneous System.*

1. INTRODUCTION

1.1 Introduce the problem

Continuously growing requirements to devices for processing, receiving and transmission of digital information cause the necessity of involving significant resources, primarily intellectual, to solve the problem of improving efficiency of means, specific algorithms and methods which implement technologies of data processing, transmission and receiving in such devices. One of the fastest growing ways to solve this problem is the coding theory, the noiseless coding in particular.

At the preliminary stages of development of telecommunication systems, containing the noiseless coding modules as inadvertent error correction means in communication channels, the optimization of these modules was an essential step. There is a promising way to reduce time given for this stage – to use the untapped resources of heterogeneous computing systems, in particular – the computing power of GPUs (Graphics Processing Unit) [1-3]. In case of sufficient parallelism of implemented calculations using GPGPU technology (General-Purpose computing on Graphics Processing Units) [4], it is possible to achieve a significant time saving compared to the same calculations, produced in series on CPU (central processing unit). The paper presents the results of this technology implementation by using an open OpenCL (Open Computing Language) standard [5] for simulation of classical convolutional encoder with Viterbi decoding.

The computing on GPU is actively developed due to promising results and their relevance, in particular by graphic card developers, providing users the means of access to internal resources of cards, tools for developing software to run on GPUs. AMD and Nvidia are definitely the leaders in the field of software implementation of GPGPU technology and their products OpenCL (Open Computing Language) and CUDA (Compute Unified Device Architecture) respectively. The leading position in terms of

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

performance and stability takes the CUDA technology from Nvidia, but the hardware and software architecture is limited to using only the Nvidia hardware solutions. Using CUDA leads to the lack of hardware independence, and under current conditions it is quite an important factor, forcing to refuse from using this technology in favor of its competitor from AMD – OpenCL. OpenCL is a completely open standard implementation of GPGPU technology, providing implementation of cross-platform (hardware-independent) applications executed in heterogeneous systems.

This task is relevant due to the fact that the means of noiseless coding are widely introduced all over the world allowing to move to new data transmission standards, to reduce the number of receiving and transmitting stations, to reduce transmitter power, to reduce the harmful effects of electromagnetic radiation on human, to improve the reliability of data transmission, and to increase data transfer speed.

The traditional approach to the noiseless coding does not allow to detect adequately, correct errors and to achieve the required level of energy efficiency; it does not fully utilize the hardware capabilities of data transmission systems.

Thus, the problem of organization of parallel computing in heterogeneous systems and the development of appropriate tools for decoder performance simulation based on the Viterbi algorithm to reduce their design time requires further research.

The problem of the parallel PRN generation today is a relevant objective and is actively being worked on [9-12]. Let us briefly consider methods of generating parallel flows of PRN.

Viterbi algorithm is the most efficient among the decoding algorithms based on the lattice diagram, since it allows to get the most realistic (MR) estimation of the transmitted code word. Standard decoder implementing Viterbi algorithm does not allow achieving the required reliability and quality, and noise immunity.

The techniques described herein and simulation tools imply the use of general-purpose computing on GPU and will allow reducing the time required to optimize the error-correcting coding system and thus reduce the time for the development and implementation of telecommunication devices.

2. METHOD

2.1 Computing performance in heterogeneous systems

One of the ways to solve the problem of reducing the time costs for the simulation process is the use of unused resources of heterogeneous computers during the calculations: in addition to the CPU resources (central processing unit), the computing resources of GPU (graphics processing unit). With the sufficient parallelizability of simulation algorithms, GPU computing is less costly in terms of time resources [3].

In [3], the problem of communication interactions of the host-computing device level was considered. In this paper, we consider the second fundamental problem – the problem of random access to memory and a high-latency of a global memory.

GPUs are theoretically far ahead of CPUs in terms of performance, but in order to "load" GPU to its highest level of performance, tasks must run in parallel in the number of computing flows, commensurately or greater than the number of streaming processors GPU. The number of stream processors in modern graphics cards is up to several thousand, so the task of providing the sufficient usage of GPU during calculations can be quite complicated, and the main "bottleneck", the diameter of which limits the performance, is typically a need to communicate with the main calculator of a heterogeneous system. Let us consider in details the architecture of heterogeneous computing systems in the context of the chosen development environment and the given task. The architecture of heterogeneous system is shown in Figure 1 from the point of view of OpenCL standard [6].

Journal of Theoretical and Applied Information Technology
31st December 2016. Vol.94. No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

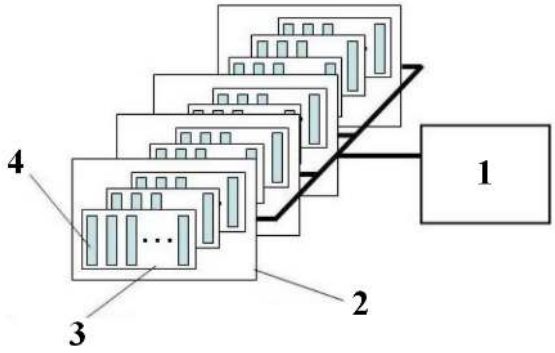


Figure 1. Architecture Of Heterogeneous Systems: 1 – Host; 2 – Computing Device; 3 – Computing Unit; 4 – Processing Element

This architecture assumes that there is one or more OpenCL devices connected to the host part (Host, receiving guests) – a device providing access services to a number of OpenCL devices, divided into several computing units with a number of processing elements. Calculations are made on the device processing elements. Commands of OpenCL application are addressed by the host to run on the processor elements inside the device. The processor elements run a single flow as SIMD (Single Instruction, Multiple Data) and SPMD (Single Program, Multiple Data) instructions.

Thus, a significant problem of GPU computing use is a set of constraints associated with the low bandwidth of the host-computing device connecting bus. This is a PCI bus for the graphics card (Peripheral Component Interconnect). For example, for a card used for the simulation in this paper, PCI Express 2.1 bus has a 5 Gb/s limit of maximum speed in one line [7-8].

The problem of PRN parallel generation is still a relevant one for today and it is actively being worked on [9-12]. Let us briefly consider methods of parallel flow generation of PRN.

As suggested in [3], the consistent PRNG (PRN generator) can be used for the parallel pseudorandom number flows generation. In applications with parallel numerical methods of statistical simulation – Monte-Carlo, additional requirements are to be met by a perfect generator, suitable for the designing of parallel PRNG:

1) scalability of generation flows, sufficient to generate the desired number of flows;
2) locality of generated flows, allowing to obtain the PRN sequences with no need for interaction with other flows;
3) mutual independence of sequences generated by different flows.

The main methods of obtaining parallel PRN flows are, in accordance with [9, 13-14]:

1.Random selection of initialization values (Random seeding).

Each flow uses the same PRNG, and different initialization values are used for different flows.

This method is applicable in case of no strict requirements for flows correlation.

2.Parameterization.

Each flow uses the same type of PRNG, with different sets of parameters for the selected flows. The example is the use of different increments c or factors a for linear congruence method $X_n = (aX_{n-1} + c) \bmod m$, where X_0 is value initializing the generator state.

The method also does not guarantee the absence of correlation flows, in addition, the number of generated flows is restricted with the set of variable parameters, i.e. scalability is not guaranteed.

1.Block Splitting.

The output sequence of generator r is divided into blocks of M length (Figure 2) [15], where M is biggest number needed for solving tasks of PRNG implementations.

To use the method these options are required:

a) Possibility to estimate in advance the top value of *M*;

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

b) Presence of an effective method for skipping values blocks by PRNG.

The result of this method depends on the existence of correlation between elements of sequence at M spaces.

2.Leapfrog.

Flows use one generator, at that each of the flows performs skipping p-1 implementations of PRNG, where p is the required number of flows (Figure 2).

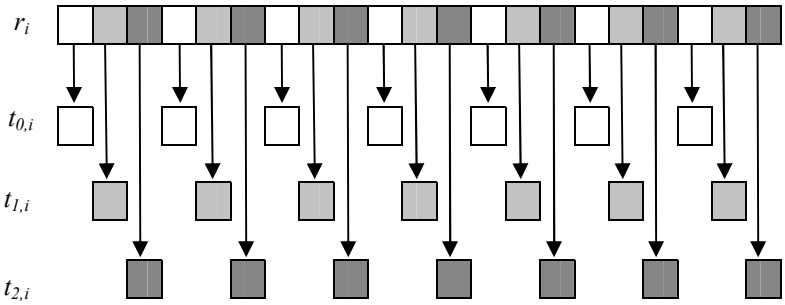


Figure 2. Parallelization With "Leapfrog" Method

The "leapfrog" parallelization method is reliable enough, before computing it is not required to estimate the amount of PRNG implementations needed to complete the task; however, it requires an efficient algorithm of generator skipping of a predetermined number of values.

The effect of flows correlation generated by PRN is not so great during the simulation of noiseless coding of low-density, in contrast to, for example, cryptography applications, so the parameterization and random initialization were chosen to parallelize PRNG.

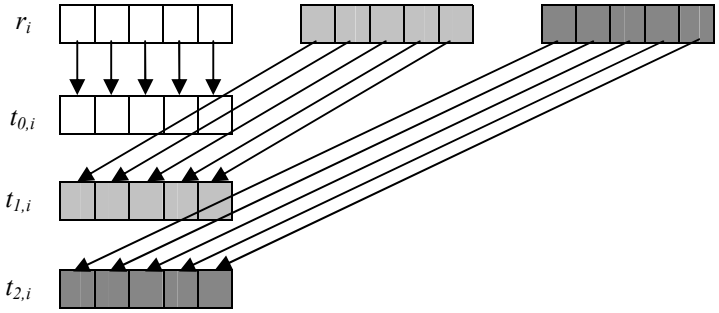


Figure 3 Parallelization With Block Splitting

Random access to the memory of the graphics accelerator and the high latency of its global sections have a significant impact on heterogeneous computing performance. Let us address the memory model, provided by OpenCL standard in this issue (Figure 4) [4].

The standard provides for the following types of memory:

1) global:
- reading and writing;
- low-speed access;

2) constant:
- access from all the processing elements (PE) to reading;

3) local:
- reading and writing one group to all PE (computing unit);
- high-speed access;

4) private:
- allocated by the compiler;
- reading and writing exclusively within PE;
- high-speed access;

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

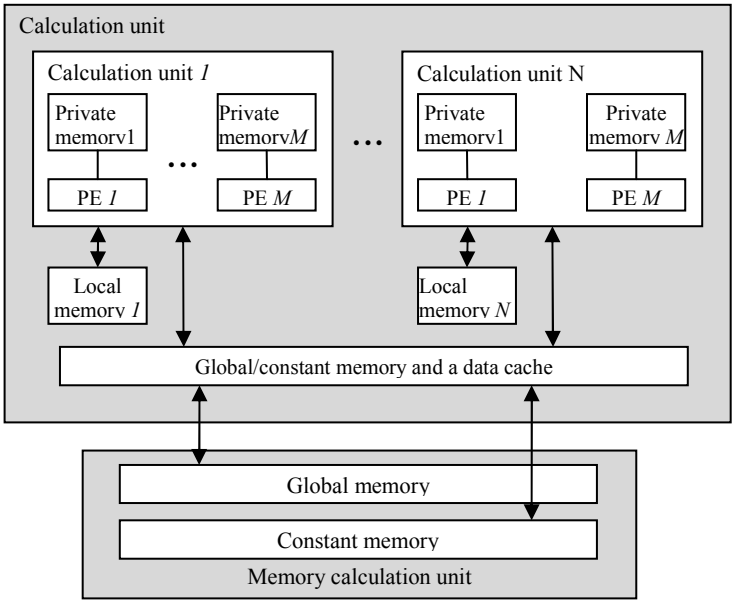


Figure 4. The Memory In Accordance With The Opencl Standard

According to this model, the main problems are:

Access to global memory itself, due to the fact it is conjugate with high latency and low speed;

Problems of random access to memory – delays during simultaneous multiple access of PEs to memory cells (collision) (Figure 5);

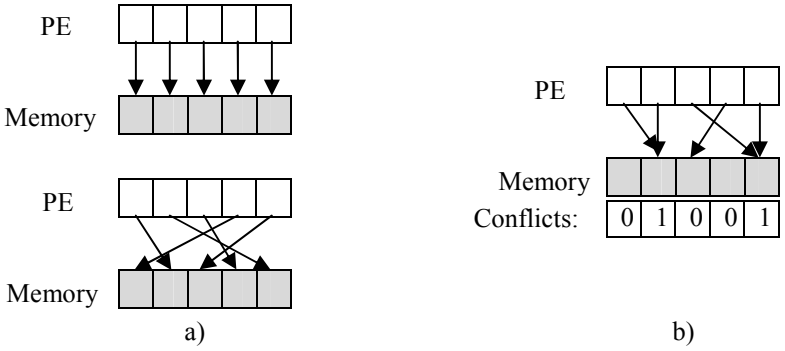


Figure 5. Conflicts Of Access To Memory: A) Conflict-Free Access; B) Collision

- problems of access synchronization within the computing device in terms of inability to access the memory of another computer unit.

Thus, when working with memory, the main principles are as follows:

- first minimize calls to slow global memory and second – to the local memory;
- use caching – copy commonly used slow memory units to a faster memory.

In papers [16-17], the caching procedure is used. The benefits of using caching procedures for some variables are shown on the diagram of Figure 6.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

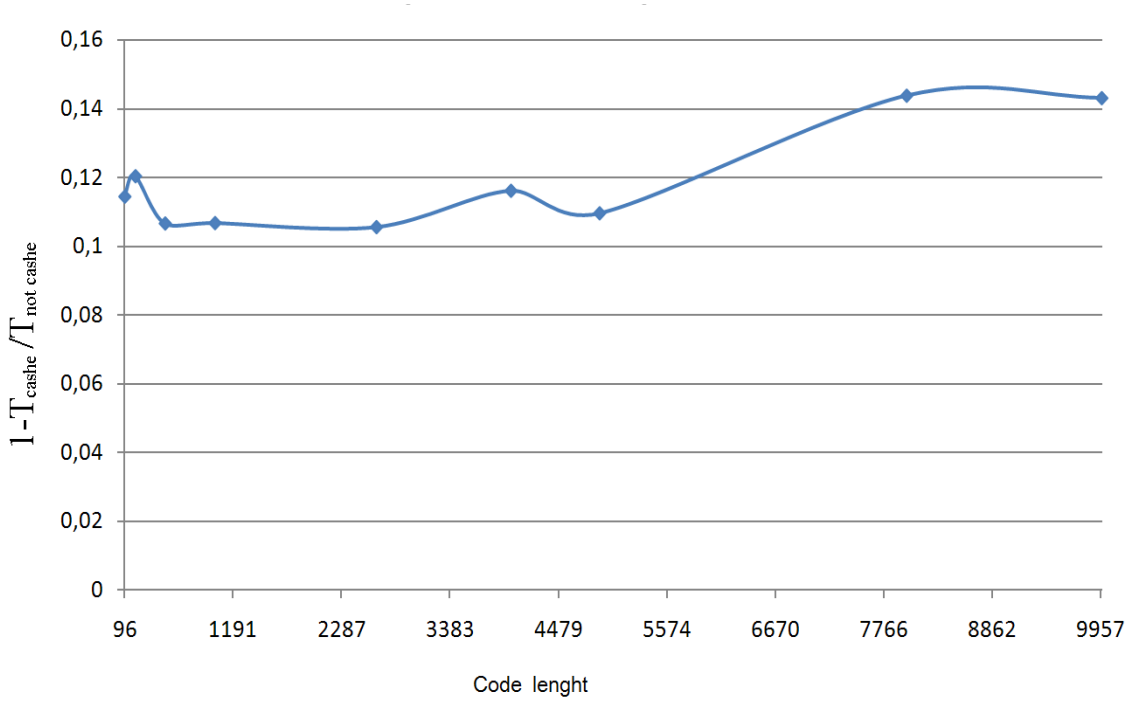


Figure 6. The Gain From The Using Caching Procedures

In Figure 6, T_{cashe} is the decoding time with the use of caching procedures and $T_{not\ cache}$ is the decoding time without caching procedures. Decoding was produced in 10 iterations and 3,408 simulations per one value of SNR (10 values of SNR were simulated: from 0.1 to 2.5 with the resolution of 0.25 dB). Graphics accelerator Radeon HD 5770 was used as a computing device (800 stream processors, 850 MHz).

2.2. Modified algorithm of parallel calculation performing

The code performed during the calculations on the GPU corresponds to counting of path metrics, selection of these survivors and updating array of metrics for the current and the previous steps [18-20]. To enable running in parallel with the necessary calculations, the operation algorithm is modified as follows:

Legend:

l – length of code constraint;

$M_0\ [2^{l-1}]$ – the array for storing the metrics of surviving paths calculated at the previous step;

$M_1\ [2^{l-1}]$ – the array of calculated metrics;

$P\ [n \cdot 2^{l-1}]$ – the array storing the number of the previous states of the survivor path state;

i – number of processing element (kernel in OpenCL terminology);

m_0 is a metric value, provided that for the previous state of i (or $i - 2^{l-2}$, if $i \geq 2^{l-2}$) there was a state $i \cdot 2$ $((i - 2^{l-2}) \cdot 2)$;

m_1 is a metric value, provided that the previous state of i or $i - 2l - 2$, there was a state $i \cdot 2 + 1$ $((i - 2^{l-2}) \cdot 2 + 1)$;

t – a number of flow metrics counting step.

Each processing element:

1. Calculates $M_1\ [i]$,

- In case of $i < 2^{l-2}$, equal to a bigger of the values $M_0\ [i \cdot 2] + m_0$ and $M_0\ [i \cdot 2 + 1] + m_1$;

$i \cdot 2$ is written in $P\ [t]$ if $M_0\ [i \cdot 2] + m_0 > M_0\ [i \cdot 2 + 1] + m_1$ and $i \cdot 2 + 1$ otherwise.

- In case of $i \geq 2^{l-2}$, according to the larger value of $M_0\ [(i - 2^{l-2}) \cdot 2] + m_0$ and $M_0\ [(i - 2^{l-2}) \cdot 2 + 1] + m_1$;

$i \cdot 2$ is written in $P\ [t]$ if $M_0\ [(i - 2^{l-2}) \cdot 2] + m_0 > M_0\ [(i - 2l - 2) \cdot 2 + 1] + m_1$ and $i \cdot 2 + 1$ otherwise case.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2. Sets of metrics are updated:

- on the first step of metrics calculation – $M_0[i] = M_1[i] = -\infty$, $M_0[0] = 0$;

- In other cases – $M_0[i] = M_1[i]$, $M_1[i] = -\infty$.

This solution is justified by the structure of the convolutional encoder trellis diagram. For example, according to the diagram of a convolutional code with a constraint equal to "4", as shown in Figure 7, it is obvious that at step t of metrics calculation, states $i \cdot 2$ and $i \cdot 2 + 1$, $0 \le i < 2^{l-2}$ on the trellis diagram are the previous ones to states $0 \le i < 2^{l-2}$, if at step $t - 1$, "0" was a new information bit, and they are the previous ones to states $2^{l-2} \le i < 2^{l-1}$, if "1" was a new information bit at step $t - 1$.

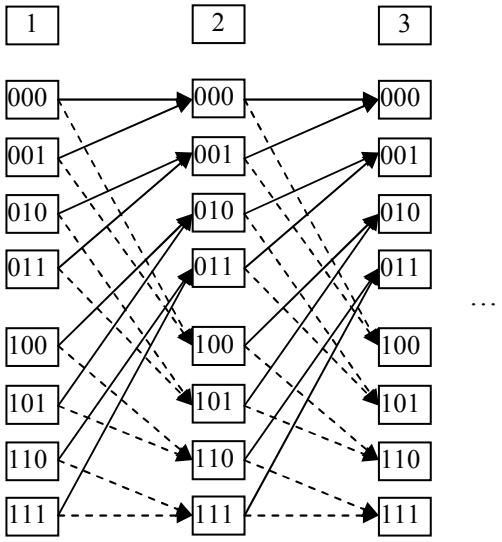


Figure 7. The Trellis Diagram Of A Convolutional Code With A Constraint Equal To "4" For Three Steps Of Counting Metrics

This fact was used for the parallelization of metrics calculation algorithm. Abbreviated code implementing the algorithm was adapted in accordance with the requirements of OpenCL and it is shown below (left – shortened code implementing the algorithm in sequence on the CPU, right – parallel version for computing on GPU):

1. Counting metrics (Kernel No. 1).

```
                                        //k1 = D/R;
                                        uint i = get_global_id(0);
            for(int i=0;i <NoS;i++)     if (i<NoS)
            {                           {
                                        if (i<NoS/2)
                                        {
if (M0[i]>-DBL_MAX)                     if  ((M0[i*2]>-  FLT_MAX)||(M0[i*2+1]>-
                  {                     FLT_MAX))
                  double m0=0;                {
                  double m1=0;                      float m0_0=0;
                  for(int                           float m0_1=0;
t=0;t<R;t++)                                        for(int t=0;t<R;t++)
                        {                           {
m0+=trfrm((rules[i][0]>>(R-1-           m0_0+=trfrm((rules0[i*2]>>(R-1-
t))&1)*rx[k*R+t];                       t))&1)*rx[R*k+t];
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
m1+=trfrm((rules[i][1]>>(R-1-
t))&1)*rx[k*R+t];
                        }
            if (M1[i/2]<M0[i]+m0)
            {
            M1[i/2]=M0[i]+m0;
            prev[i/2][k]=i;
            }
```

```
if (M1[i/2+NoS/2]<M0[i]+m1)
                    {
      M1[i/2+NoS/2]=M0[i]+m1;
      prev[i/2+NoS/2][k]=i;
                    }
                  }
            }
```

```
m0_1+=trfrm((rules0[i*2+1]>>(R-1-
t))&1)*rx[R*k+t];
                  }
      if
((M0[i*2]+m0_0)>(M0[i*2+1]+m0_1))
      {
      M1[i]=M0[i*2]+m0_0;
      prev[i*k1+k]=i*2;
      }
      else
      {
      M1[i]=M0[i*2+1]+m0_1;
      prev[i*k1+k]=i*2+1;
      }
      }
}
else
{
      if ((M0[(i-NoS/2)*2]>-FLT_MAX)||
       (M0[(i- NoS/2)*2+1]>-FLT_MAX))
      {
            float m1_0=0;
            float m1_1=0;
            for(int t=0;t<R;t++)
            {
m1_0+=trfrm((rules1[(i-NoS/2)*2]>>
        (R-1-t))&1)*rx[R*k+t];
m1_1+=trfrm((rules1[(i-NoS/2)*2+1]>>
        (R-1-t))&1)*rx[R*k+t];
            }
if ((M0[(i-NoS/2)*2]+m1_0)>
(M0[(i-NoS/2)*2+1]+m1_1))
            {
            M1[i]=M0[(i-
NoS/2)*2]+m1_0;
            prev[i*k1+k]=(i-NoS/2)*2;
            }
            else
            {
            M1[i]=M0[(i-
NoS/2)*2+1]+m1_1;
            prev[i*k1+k]=(i-
NoS/2)*2+1;
            }
      }
}
}
}
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2. Updating arrays of metrics (Kernel No. 2).

```
                                        uint i = get_global_id(0);
        for(int i=0;i<NoS;i++)          if (i<NoS)
        {                               {
                                        if(k==k1-1)  {
                                                     M0[i]=-FLT_MAX;
                                                     if (i==0) {M0[0]=0;}
                                                     }
              M0[i]=M1[i];              else M0[i]=M1[i];
              M1[i]=-DBL_MAX;           M1[i]=-FLT_MAX;
        }                               }
```

1. Counting metrics with synchronization (Kernel No. 3).

```
                                        //k1 = D/R;
                                        uint i = get_global_id(0);
                                        if (i<NoS)
                                        for(int k=0;k<k1;k++)
  for(int i=0;i<NoS;i++)                {
  {                                     if(k==0)    {
        M0[i]=M1[i]=-DBL_MAX;                        M0[i]=M1[i]=-FLT_MAX;
        for(int k=0;k<D/R;k++)                       if (i==0) { M0[0]=0; }
        prev[i][k]=-1;                               }
  }
  M0[0]=0;
  for(int k=0;k<size/R;k++)
  {
        for(int i=0;i<NoS;i++)
        {
                                        if (i<NoS/2)
                                        {. . . }
if (M0[i]>-DBL_MAX)                     else
              {. . . }                  {. . .}
        }
        for(int i=0;i<NoS;i++)          barrier(CLK_GLOBAL_MEM_FENCE);
        {
              M0[i]=M1[i];              M0[i]=M1[i];
              M1[i]=-DBL_MAX;           M1[i]=-FLT_MAX;
        }                               }
  }
```

3. RESULTS

In the provided short listing on the right – the code, that runs separately by GPU 2^{l-1} cores [21-23]. Code `get_global_id(0)` provides the identification of kernel executing the code instance. Thus, in the pseudo code to the left, cycles `for (int i=0; i < NoS; i++)`, where `NoS` = 2^{l-1} run parallel on GPU. The number of running flows Th_g (`globalThreads`) and the size of a working group Th_l (`localThreads`) for running on the GPU 2^{l-1} instances of code are defined by the expression:

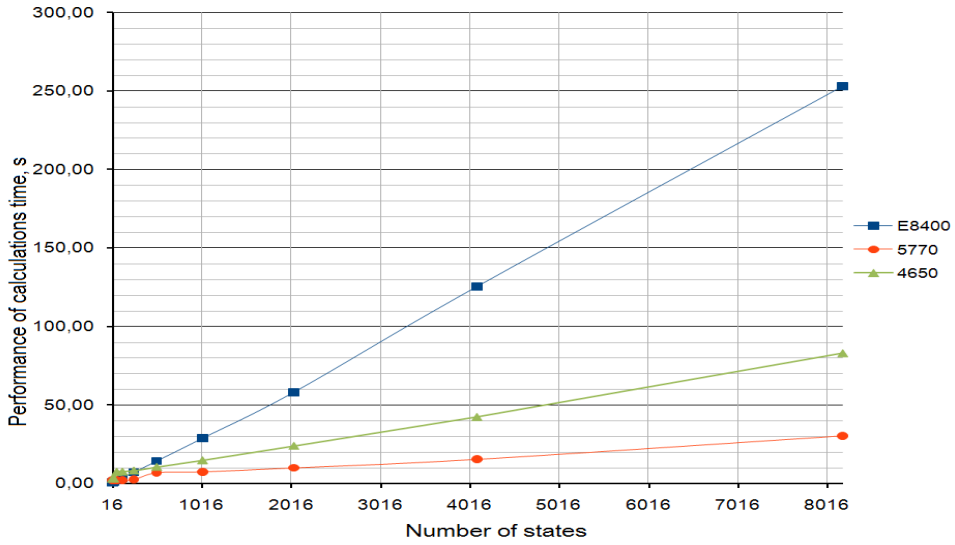
$$\begin{cases} 2^{l-1} > Th_{l\max}, Th_g = 2^{l-1} + Th_{l\max} - \left[2^{l-1} \bmod (Th_{l\max})\right], Th_l = Th_{l\max}; \\ 2^{l-1} \le Th_{l\max}, Th_g = Th_l = 2^{l-1}. \end{cases} \quad (1)$$

where $Th_{l\max}$ is the maximum size of a one-dimensional local workgroup (`CL_KERNEL_WORK_GROUP_SIZE`).

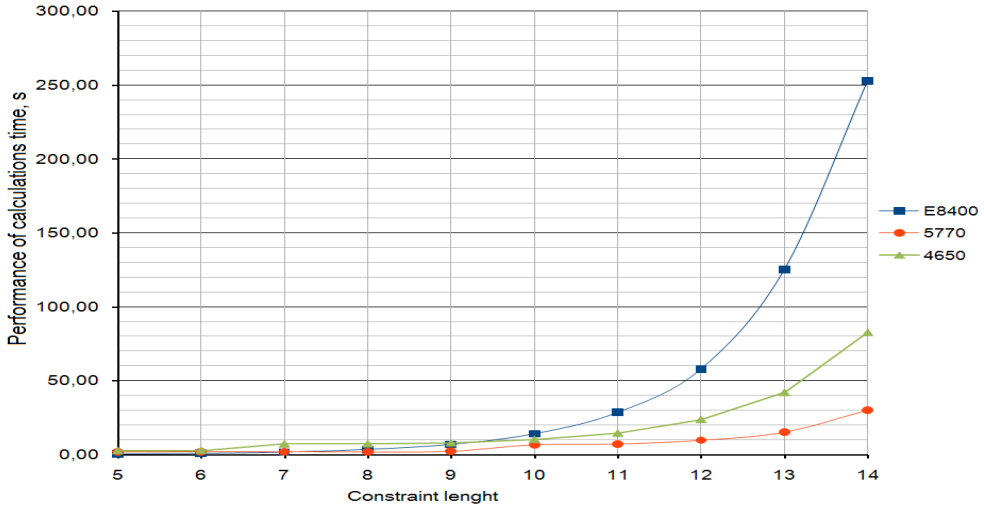
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

If the $2^{l-1} \leq Th_{lmax}$, is performed only by Kernel No. 3, or sequentially for *D/R* times in one after another Kernel No. 1 and Kernel No. 2, starting from Kernel No. 1 are performed, where D is the length of decoded packet and R is a value of inverse code rate.

In these modes, calculations were performed on a AMD Radeon HD 5770 graphic card (800 flow processors with a frequency of 850 MHz), AMD Radeon HD 4650 (320 flow processors with a frequency of 600 MHz) and CPU – Intel Core 2 Duo E8400 in single-flow mode with a frequency of 3.6 GHz. The simulation was performed at 100 iterations of coding, decoding and noise pollution of packet with the length of 1024 bit, with the noise level of 1; 1.5; 2; 2.5 dB with the code rate of 0.5. Diagrams of time dependency on the number of states during the simulation and the constraint length are shown in Figure 8 a) and b).



a)



b)

Figure 8. Simulation Of Noiseless Convolutional Coding System With Viterbi Decoding: A) Dependence Of The Computing Time On The Number Of Encoder States; B) Dependence Of Computing Time On The Constraint Length

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Applying this model we managed to reduce the payment execution time. Figure 9 shows the time benefit of applying the parallelization scheme described above, depending on the code length.

4. DISCUSSION

Curves on Figure 9: E8400 – corresponds to the time required for modeling sequential computing on CPU Intel Core 2 Duo E8400; 5770 – costs of simulation using parallel computing on GPU Radeon HD 5770; 4650 – on the GPU Radeon HD 4650 respectively. For GPU Radeon HD 5770, time gain compared to the CPU E8400 becomes apparent at $2^{l-1} = 128$; for the Radeon HD 4650 – at $2^{l-1} = 512$. For clarity, Figure 8 shows the dependency graphs of the time ratio of calculation performing with the calculations of the CPU to the GPU on the number of convolutional encoder states.

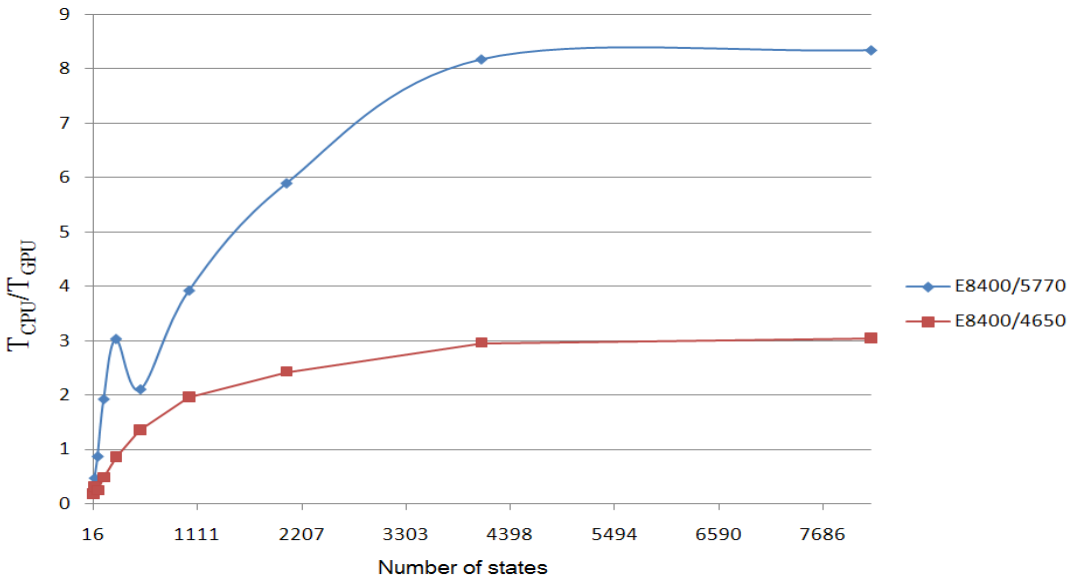


Figure 9. Relative Time Gain Of Gpu Computing

The dip in the curve "E8400/5770" at the point corresponding to the value of the $2^{l-1} = 512$ is associated with the size limitations of "working group" during the calculations on Radeon HD 5770. The driver of graphic card limits the size of the value to 256. At $2^{l-1} \leq 256$ kernel No. 3 runs, corresponding to the algorithm with synchronization of the local group, which provides the possibility to perform metrics calculation of the paths for each step of initializing kernel only at the first step and provides the possibility to update sets of metrics at the time of guaranteed completion of the metrics counting within a single kernel, resulting in reduced communication costs of host-kernel levels.

Thus, for small length values (N=96), the time gain is 80% and decreases to 60% at N=204, and 41% when N=273. Then there is a stable site with an average gain value of 21%, when the code length is in range from N=504 to 3000.

The scope of this method is limited by the condition $Q < Max_{sim}$. It should be noted that basically this method can be implemented in heterogeneous systems with multiprocessor CPU, based with the full load of all CPU cores.

5. CONCLUSION

Thus, the main problems of noiseless codec simulation in heterogeneous systems are problems of low bandwidth of connecting bus of host-computing device, the problem of random access to memory and high-latency of global memory. As part of the solution of the first problem, it is proposed to follow the principles of minimizing the levels of host-computing device communications, and as for the problem of access to memory – to

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

follow the principles of minimizing costly accesses to slower memory and applying caching procedures.

The effect of a low flow capacity of a global and local memory of GPU on computing performance of automated simulation tools of decoders can be reduced by the use of developed computational schemes by minimizing request to these memory sections and applying caching procedures. Experimentally estimated computing performance increase for code length of 96... 9972 comprises 11% on average.

The problem of reducing the computing performance as the result of interactions of two communication levels of CPU-GPU. To limit communication interactions, the model of noise source should be implemented at the level of processing elements of GPU. The use of the developed simulation method of the noise source provides sufficient accuracy of the results and the increase in computing performance by reducing the number of requests to an external status register of the pseudorandom number generator. Experimentally estimated computing performance increase reaches 30%. The results of the conducted research can be applied for decoder simulation used in the wireless, interference-proof networks, such as WLAN, WPAN, WMAN.

The results of GPGPU technology application give reason to believe this approach to be promising to solve the problem of reducing the time spent on resource-intensive simulations of error-correcting codec characteristics even in the case of classical convolutional codes decoded by Viterbi algorithm.

REFERENCES:

[1] Bashkirov, A.V., Klimov, A.I., Muratov, A.V., Naumenko, Yu.S., & Tsymbalyuk, V.S. (2013). Perspektivy modelirovaniya parametrov algoritmov pomekhoustoychivogo kodirovaniya s vysokoy stepen'yu parallelizma pri pomoshchi apparatnoy platformy na baze GPU [Outlooks for Simulation of Parameters of Noiseless Coding Algorithms with a High Degree of Parallelism with the Help of Hardware Platform Based on GPU]. *Radiotekhnika, 12*, 26-29.

[2] Naumenko, Yu.S. (2014). Problemy modelirovaniya pomekhoustoychivykh kodekov v geterogennykh sistemakh [Problems of Noiseless Codecs Simulation in Heterogeneous Systems]. *Radiotekhnika, 3*, 80-82.

[3] Naumenko, Yu.S. (2014). Massivnye parallel'nye vychisleniya v geterogennykh sistemakh pri modelirovanii nizkoplotnosnykh kodekov [Massive Parallel Computations in Heterogeneous Systems during Simulation of Low-Density Codecs]. *Radiotekhnika, 6*, 43-46.

[4] Boreskov, A.V., & Kharlamov, A.A. (2010). *Osnovy raboty s tekhnologiey CUDA* [Basic Operation with CUDA Technology]. Moscow: DMK Press.

[5] Khronos OpenCL Working Group. (2013, November 14). *The OpenCL Specification*. Version: 2.0. Document Revision: 19.

[6] Zhmurov, A.A., Varsegov, V.A., Trifonov, S.V., Kholodov, Ya.A., & Kholodov, A.S. (2011). Effektivnye generatory psevdosluchaynykh chisel pri molekulyarnom modelirovanii na videokartakh [Effective Pseudorandom Number Generators for Molecular Simulation on Graphic Cards]. *Komp'yuternye issledovaniya i modelirovanie, 3*(3), 287-308.

[7] Bauke, H., & Mertens, S. (2007). Random Numbers for Large Scale Distributed Monte Carlo Simulations. *Physical Review E, 75*(6).

[8] Manssen, M., Weigel, M., & Hartmann, A. (2012). Random Number Generators for Massively Parallel Simulations on GPU. *Eur. Phys. J. Special Topics, 210*(1), 53-71.

[9] Giles, M. (2011). Approximating the Erfinv Function. In W.W. Hwu (Ed.), *GPU Computing Gems* (Vol. 2) (pp. 109-116). Burlington: Morgan Kaufman.

[10] Bradley, T., Toit, J., Giles, M., Tong, R., & Woodhams, P. (2010). Parallelisation Techniques for Random Number Generators. In W.W. Hwu (Ed.), *GPU Computing Gems* (Vol. 1). Morgan Kaufmann.

[11] Blahut, R. (1986). *Teoriya i praktika kodov, kontroliruyushchikh oshibki* [Theory and Practice of Error Control Codes] (I.I. Grushko, & V.M. Blinovskiy, Trans.). Moscow: Mir.

[12] Bauke, H. (2014, August 5). *Tina's Random Number Generator Library*. Version 4.17. Retrieved August 1, 2016, from http://numbercrunch.de/trng/.

[13] Blahut, R. (1989). *Bystrye algoritmy tsifrovoy obrabotki signalov* [Fast Algorithms for Digital Signal Processing] (Trans. from English). Moscow: Mir.

[14] Barash, L.Yu., & Schur, L.N. (2013). O generatsii parallel'nykh potokov

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

psevdosluchaynykh chisel [About Generation of Parallel Flows of Pseudorandom Numbers]. *Programmnaya inzheneriya, 1*, 24-32.

[15] Falcao, G., Sousa, L., & Silva, V. (2009). How GPUs Can Outperform ASICs for Fast LDPC Decoding. In *ICS '09: Proceedings of the 23rd international conference on Supercomputing* (pp. 390-399). New York: ACM.

[16] Bykov, V.V. (1971). *Tsifrovoe modelirovanie v statisticheskoy radiotekhnike* [Digital Simulations in Statistical Radio]. Moscow: Sovetskoe radio.

[17] Komashinskiy, V.I., & Maksimov, A.V. (2007). *Sistemy podvizhnoy radiosvyazi s paketnoy peredachey informatsii. Osnovy modelirovaniya* [Mobile Radio Systems with Packet Data Transmission. Basics of modeling]. Moscow: Goryachaya liniya – Telekom.

[18] Savinkov, A.Yu. (2006). *Avtomatizatsiya proektirovaniya sistem tsifrovoy obrabotki signala na osnove integrirovannoy sredy imitatsionnogo modelirovaniya i optimizatsii: Dis. d-ra tekhn. nauk* [Computer-Aided Design of Digital Signal Processing Systems Based on Integrated Environment Simulation and Optimization (Doctoral Thesis in Technical Sciences)]. Voronezh.

[19] Tikhonov, V.I., & Kharisov, V.N. (1991). *Statisticheskiy analiz i sintez radiotekhnicheskikh ustroystv i system* [Statistical Analysis and Synthesis of radio Engineering Devices and Systems]. Moscow: Radio i svyaz'.

[20] MacKay, D.J.C., & Davey, M.C. (2000). Evaluation of Gallager Codes for Short BlockLength and High Rate Applications. In B. Marcus, & J. Rosenthal (Eds.), *Codes, Systems and Graphical Models*: Vol. 123 of IMA Volumes in Mathematics and its Applications (pp. 113-130). New York: Springer.