GPGPU accelerated optimization method of Interconnection Network Topology

VLADIMÍR SILÁDI[1], LADISLAV HURAJ[2], MICHAL POVINSKÝ[1]
[1]Department of Computer Science
University of Matej Bel
Tajovskeho 40, 97401 Banska Bystrica
SLOVAK REPUBLIC
vladimir.siladi@umb.sk, antihutka@gmail.com
[2]Department of Applied Informatics and Mathematics
University of SS. Cyril and Methodius in Trnava
Nám. J. Herdu 2, 917 01 Trnava
SLOVAK REPUBLIC
ladislav.huraj@ucm.sk

Abstract: The optimization of the irregular connection network of the multiprocessor systems with the distributed memory is the NP complete problem which is generally compute-intensive process. Graphics processing units provide a large computational power at a very low price allowing the fine-grained parallelism. This work investigates the use of the GPU in the parallelisation of the optimal irregular network configurations up to 128 processors. We used a modified hill climbing optimization technique for finding better solutions for interconnection network topology. Using NVIDIA's Compute Unified Device Architecture, our implementation achieves a processing speed up of 1.71 to 7.96 times over a sequential Central Processing Unit approach and it is comparable to existing approaches.

Key-Words: interconnection networks, irregular topology, optimization algorithm, GPGPU

1 Introduction

Graphics Processing Units (GPUs) are widely used among developers and researchers as accelerators for applications outside the domain of traditional computer graphics. In particular, GPUs have become a viable parallel accelerator for scientific computing with low investment in the necessary hardware [1,2]. After the establishment of the first official version of the application interface for general computing on graphics cards in 2007, the usage of GPUs for acceleration of general computations has expanded massively. This trend largely results from the great improvements in GPU programmability (i.e. GPGPU or use of GPUs for general purpose applications) [3]. CUDA (Compute Unified Device Architecture) is a parallel computing architecture developed by NVIDIA for graphics processing which presents to the programmer a fairly generic abstraction of a many-core architecture supporting fine-grained parallelism. CUDA and the GPU therefore provide massive, general purpose parallel computation resources with the potential for speedups of data processing.

In our article we focus on utilization of GPUs in the area of the multiprocessor computer architecture design. Building of multiprocessor system with hundreds of processors is achievable in the present day. The interconnection network is an essential element of designed multiprocessor systems and it is critical to determine its performance [4]. The optimization of the irregular connection network of the multiprocessor systems with a distributed memory is included among the NP-complete problems.

Principles and practices of interconnection networks have been studied in the past, e.g. [6]. To provide low latency and high bandwidth communication in interconnection networks, many researches have been carried out to optimize the network in various approaches such as designing new network topologies, e.g. [5,7,8].

This work details the GPU parallelisation of the optimization process of the irregular connection network of the multiprocessor systems with the distributed memory.

The rest of the paper is organized as follows. First we briefly review the CUDA programming model and we introduce the background of interconnection network problem in section 2.

Section 3 describes the algorithms and their mapping on the GPU. Section 4 shows our experimental results. The conclusion comes in section 5 with an outlook to further work.

2 Background

This section provides the basic background for the further discussions in the later sections of this article. Section 2.1 details GPGPU principles and the architecture of CUDA. The concept of interconnection network, graph representation as well as the basic optimization techniques are introduced in section 2.2.

2.1 General Purpose computing on Graphics Processing Units

Graphic cards, especially GPUs, represent quite new possibility for high performance computing (HPC). On the other hand, using of GPGPU is purposeful only if the larger amounts of data are processed.

Parallel computing architecture developed by NVIDIA named CUDA allows programmers to use the graphic cards for parallel programming. The Compute Unified Device Architecture (CUDA) allows developers to use the C programming language for the development of general-purpose applications using fine-grain parallelism. A simple extension to C has invoked that more non-graphics developers port their existing applications to CUDA. CUDA consists of a runtime library and an expanded version of C. CUDA gives developers access to the native instruction set and memory of the parallel computational elements in CUDA GPUs. It includes the CUDA Instruction Set Architecture (ISA) and the parallel compute engine in the GPU [10].

To support a heterogeneous system architecture combining CPU and GPU, a single source program contains both the host (CPU) code and the device (GPU) code which are automatically separated and compiled by the CUDA compiler tool chain.

CUDA is based on the notion of a core code called kernel function, which is a single routine that is invoked concurrently across many thread instances; a software controlled scratchpad, which CUDA calls the "shared memory", in a Single Instruction Multiple Data (SIMD) fashion for each SIMD core; and barrier synchronization. Each GPU thread in the thread block processes as a fully independent, scalar, and can execute arbitrary code and access arbitrary addresses. Moreover, each of the GPU threads is given a unique ID that is accessible within the kernel through the built-in threadIdx variable [11].

2.2 Interconnection Networks

Basically, an interconnection network is a system formed by nodes and links among the nodes. There are two classes of interconnection network topologies, static and dynamic. Networks with static linking, also called direct networks, are applied in networks where the communications among nodes are known, or can be estimated. The links are permanent and unchanging over time. Otherwise, in the dynamic topology there are changes of connections among processors. The algorithm proposed in this paper deals only with static network topology.

An interconnection network of a multiprocessor architecture can be represented by an undirected graph. Vertices of the graph represent computing nodes and edges of the graph represent communication links between the nodes. The graph used for representation of the topology is undirected - messages may pass between connected processors in both directions. Maximum valence l of vertices is given by number of communication links of processors. Two vertices have to have valence $l - 1$ because corresponding processors of the configuration have to be connected to the control processor. Such a graph is called the topological structure of the interconnection network.

Some interconnection network topologies are designed and some borrow from nature. For instance hypercubes, complete binary trees, butterflies and torus networks are some of the designed architectures. On the other hand, grids, hexagonal networks, honeycomb networks and diamond networks are called natural architectures [9].

One of the important criteria for optimizing the network configuration is the maximum distance m as well as the average distance p, which must be overcome between the processors by the message. The average inter-process distance is equivalent to the average number of links through which the message has to go on the path from any source processor to the target processor.

Topology of the network can be formed as regular or irregular. The regular topology is characterized by its regularity; structure of irregular topology is not regular. While minimizing the distance between processors, irregular topologies that optimize the above mentioned criteria should be formed. Such architectures provide a better alternative to the extensive regular topologies for parallel solutions of real computing problems.

The optimization effort lies in searching a graph with minimal diameter, because the diameter represents the largest distance which the message has to traverse. The maximum distance can be formally described as:

$$m = max\ d(u, v) \tag{1}$$

and average distance as:

$$p = \frac{\sum_u \sum_v d(u,v)}{n(n-1)} \tag{2}$$

where d distance between vertices, u and v are vertices, and n is the number of vertices. The graph's diameter is the largest number of vertices which must be traversed in order to travel from one vertex to another when paths which backtrack, detour, or loop are excluded from consideration.

The main goal of the work is to design an optimization algorithm of static interconnection network among processors which can be accelerated with graphics cards and to compare the sequential and parallel implementation of the algorithm. Mono-processor computer is fundamentally suitable for sequential (consecutive) data processing. On the other hand, the solving of computationally intensive tasks as the optimal irregular network configuration requires increasing of performance, which can be done through parallel processing of the data. And just GPGPU represents new technical means to manage time-critical computing tasks.

3 Optimal irregular configurations

The design of irregular interconnection networks requires finding an optimal irregular configuration over a very large search space, the space of the order of 10^{47} for 32 processors and 10^{369} for 128 processors [12]. The size of the solution space is too large to use the exhaustive search algorithm, so a *hill climbing modified algorithm* was used based on finding better solutions lying near to the best actual solution. Moreover, the CPU based approach has been significantly modified into parallel approach to suit the GPU computing paradigm.

The hill climbing algorithm as a mathematical optimization technique, is an iterative algorithm used as a local search technique. The algorithm starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, an incremental change is made to the new solution, being repeated until no further improvements can be found. In generally, hill climbing algorithm strives to escape local optima by accepting inferior solutions in an effort to reach a global optimum.

As a starting point the undirected complete graph was chosen. Consequently, the randomly chosen edges are removed respectively. The number of connections of vertices connected by edge is taken into account when the edge is removing from the graph as well as the number of paths passing through the edge and the length of the longest path passing through the edge. There is monitoring of average distance and diameter during edges' removing. Using of such monitoring ensures that the graph is all the time connected. The maximum tolerable diameter as well as initial average distance is predetermined. In case of multiple failure of edges' removing, the value of initial average distance increases.

If more than one vertex is less than the set degree after edges' removing, an edge is added between these vertices. After reaching of the required number of edges, an optimization of the pairs' exchange is made. The optimization of the pairs' exchange consists of random selection of a group of four vertices A, B, C, D, connected by edges A-B C-D. Edges between them are swapped to A-D, B-C and the maximum and the average distance is determined.

If the distance is better than the distance before the change, the change is kept, if not, the state is returned to the state before the change. Search continues until a predefined amount of trials without improvement is performed. If the solution is better than the existing one, it is marked as the best solution. Otherwise, the existing best solution is kept. Then the edges between vertices with a maximum distance are added and algorithm is repeated till predefined number of repetitions. The number of edges added at each repetition decreases.

The Floyd-Warshall algorithm to evaluate the average and maximum distance between vertices in the graph was used; the Floyd-Warshall algorithm is an algorithm designed to find the shortest paths between all pairs of vertices in graph [13].

In order to take advantage of the GPU, the hill climbing algorithm was reinterpreted in a data parallel fashion. The reinterpretation has a significant impact on the performance of the final GPU implementation.

Furthermore, the proposed algorithm consists of matrix operations which allow faster processing on graphics cards.

3.1 Implementation of the optimization algorithms on graphics cards

The above described algorithm was identified into several algorithmic building blocks and implemented them as separate functions.

Using the kernel function *floydwarshall_kernel* the Floyd-Warshall algorithm runs on matrix and the distances and paths between vertices are calculated. Modified kernel function *floydwarshall_kernel_nopaths* only counts the distances. Kernel *floydwarshall_kernel_paths* calculates the next vertex's path for each pair of vertices from the data calculated by Floyd-Warshall algorithm. Kernel *floydwarshall_kernel_longest* computes the length of the longest path through each edge. Kernel *floydwarshall_kernel_numpaths* determines the number of paths through each edge. Kernel *graph_count_edges_kernel* calculates the degree of each node. Kernel function *graph_remove_edges_kernel* randomly removes edges from the graph. Kernel function *graph_add_edges_kernel* randomly adds edges to the graph.

There are several supporting matrix kernel functions to perform basic matrix operations: *matica_set_kernel*, *matica_diag_kernel*, *matica_co-py_ones_kernel*, *matica_sum_max_kernel*.

An example of parallelization of Floyd-Warshall algorithm is illustrated on the flowcharts, Fig. 2 a,b.

4 Experimental Results

Testing was carried out on a single CPU personal computer with a single GPU. The experiment was designed to test the algorithm in terms of speeding up calculations using the graphics card compared to the CPU. The experiment was realized in an NVIDIA CUDA environment on a graphics card GeForce 9800 GT and on processor Intel Core2 Duo E7400. The program was written in C language.

The number of nodes in the graph was designated to value 32, 53, 64 and 128; 4 edges for each node. In Table 1 are described the parameters of detected networks – the maximum and the average distance of nodes for all selected numbers of nodes as well as running time on the CPU and GPU in minutes.

The Speed up column describes the acceleration of calculation on the graphics processor compared with the CPU for the topology. The acceleration was achieved for each number of nodes, the higher number of nodes, the higher efficiency factor.

Table 2 and Figure 1 compare the obtained results of our experiment with the theoretical minimum values and with six other algorithms for interconnection network topology optimization. The algorithms are as follows [12]:

- Alg. 1. backtracking search algorithm,
- Alg. 2. the classical genetic algorithm,
- Alg. 3. the genetic algorithm with mutation operator,
- Alg. 4. the genetic algorithm with simulated annealing,
- Alg. 5. the genetic algorithm with crossover operator,
- Alg. 6. the genetic algorithm with crossover operator with limited local search,
- Alg. 7. our algorithm performed on GPU.

Figure 3 shows the generated interconnection network topology with the shortest achieved average distance among nodes for the value of 53 nodes.

Table 1: Experimental results

# nodes	Distance		Time (min)		Speed up
	Max.	Avg.	CPU	GPU	
32	3	2.3548	24	14	1.71
53	4	2.7235	82	36	2.27
64	4	2.8918	140	32	4.34
128	5	3.5153	1082	136	7.96

Table 2: Comparison of different algorithms

	Number of vertices							
Alg.	32		53		64		128	
Theor.	3	2.291	4	2.578	4	2.824	5	3.409
Alg. 1	4	2.31	4	2.76	4	2.92	5	3.58
Alg. 2	4	2.30	4	2.72	5	2.92	6	
Alg. 3	4	2.47						
Alg. 4	3	2.30	4	2.693	4	2.88	5	3.52
Alg. 5	3	2.297	4	2.693	4	2.878	5	3.511
Alg. 6	3	2.297	4	2.685	4	2.873	5	3.499
Alg. 7	3	2.355	4	2.727	4	2.898	5	3.515

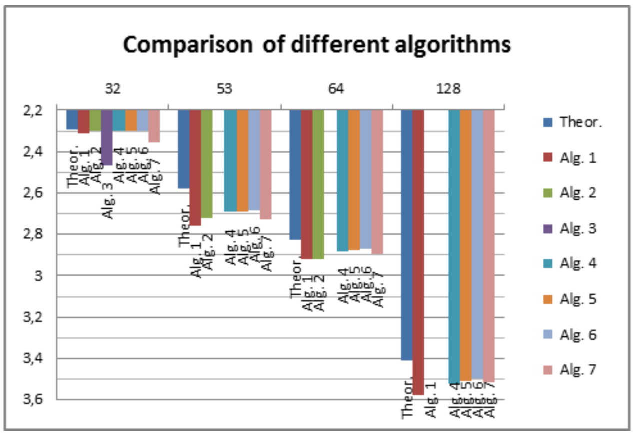

Fig. 1 Graph of comparison of different algorithms

5 Conclusion
The experiment was designed to test the algorithm in terms of speeding up the calculations through the graphics card compared to the CPU implementation for the multiprocessor computer architecture design. The experiment was realized in an NVIDIA CUDA environment, written in C language; 32, 53, 64 and 128 number of nodes; 4 edges for each node. Results of the experiment of the proposed algorithm were compared to existing published algorithms. The proposed algorithm is based on hill climbing algorithm and its results are mainly for larger number of nodes comparable to existing algorithms, Fig. 1. The implementation of optimization method through the GPGPU shows one of the feasible ways to improve the processing.

Moreover, the perspective for additional acceleration can be assumed in the handling of the program for more efficient memory access. Running of the program was successful even for 4 245 nodes. Graphs with larger amount of vertices could be processed by using GPU with larger memory, by transferring unused matrixes in system memory, or by processing of matrixes in parts.

Furthermore, another possibility to escape from a local optimum for the algorithm will be tempt [14,15]. An open issue is an adapting of the algorithm for parallel processing on multiple graphics cards.

Acknowledgements: This work was supported in part by the University of SS. Cyril and Methodius in Trnava under project FPPV-13-2012.

References:

[1] Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krger, J., Lefohn, A.E., Purcell, T.J.: A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum* 26(1), 80, 2007.

[2] Huraj, L., Siládi, V, Siláči, J.: Comparison of Design and Performance of Snow Cover Computing on GPUs and Multi-core processors. In: *WSEAS Transactions on Information Science and Applications*, Issue 10, Volume 7, October 2010, pp. 1284-1294.

[3] T.D. Han, T. S. Abdelrahman. hiCUDA: High-Level GPGPU Programming. In: *IEEE Transactions on Parallel and Distributed Systems*, 31 Mar. 2010.

[4] V. Puente, J. Gregorio, and R. Beivide. SICOSYS: an integrated framework for studying interconnection network performance in multiprocessor systems. In *Parallel, Distributed and Network-based Processing*, 2002. Proceedings. 10th Euromicro Workshop on, pages 15–22.

[5] W. J. Dally. Express Cubes: Improving the Performance of k-ary n-cube Interconnection Networks. *IEEE Trans. on Computers*, 40(9):1016-1023, 1991.

[6] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Pub., San Francisco, CA, 2003.

[7] J. Kim, J. Balfour, and W. J. Dally. Flatterned Butterfly Topology for On-Chip Networks. *Proc. of the 40th Int. Sym. on Microarchitecture*, pp. 172-182, 2007.

[8] Yuanping Mu and Keqiu Li. Extended Folded Cube: A Improved Hierarchical Interconnection Network. In: *Parallel Architectures, Algorithms and Programming (PAAP), 2011 Fourth International Symposium on* Parallel Architectures, Algorithms and Programming, pp.77-81, Dec. 2011

[9] Indra Rajasingh, Bharati Rajan, and S.Teresa Arockiamary: Irregular Total Labeling of Grid Networks. *Journal of Computer and Mathematical Sciences*, Vol. 2, Issue 6, 31 December, 2011, pp. 780-898.

[10] Wang, Y., Feng, Z., Guo, H., Ch. Yang, He, Y., “Scene Recognition Acceleration Using CUDA and OpenMP,” pp.1422-1425, *First IEEE International Conference on Information Science and Engineering*, 2009.

[11] “NVIDIA CUDA Programming Guide v2.0,” Availabe on: http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf , July 2008.

[12] Colin J. Burgess, Alan G. Chalmers, Optimisation of irregular multiprocessor computer architectures using genetic algorithms. *Annals of Operations Research*, 86, ISSN 0254-5330, pp. 239–257. February 1999.

[13] Wr04, Wróblewski P. 2004. *Algoritmy, datové štruktury a programovací techniky*. Brno: Computer Press, 2004. 351 s. ISBN 80-251-0343-9.

[14] OLEJ, V.: *Modelovanie ekonomických procesov na báze výpočtovej inteligencie*. Hradec Králové: M&V, 2003, 160 pp, ISBN 80-903024-9-1.

[15] Tanuska, Pavol - Skripcak, Tomas: The Proposal of Functional User Requirements Generation. In: *ICCRD 2011 : 3rd International Conference on Computer Research and Development*. China, IEEE, 2011, ISBN 978-1-61284-840-2. - pp. 39-42

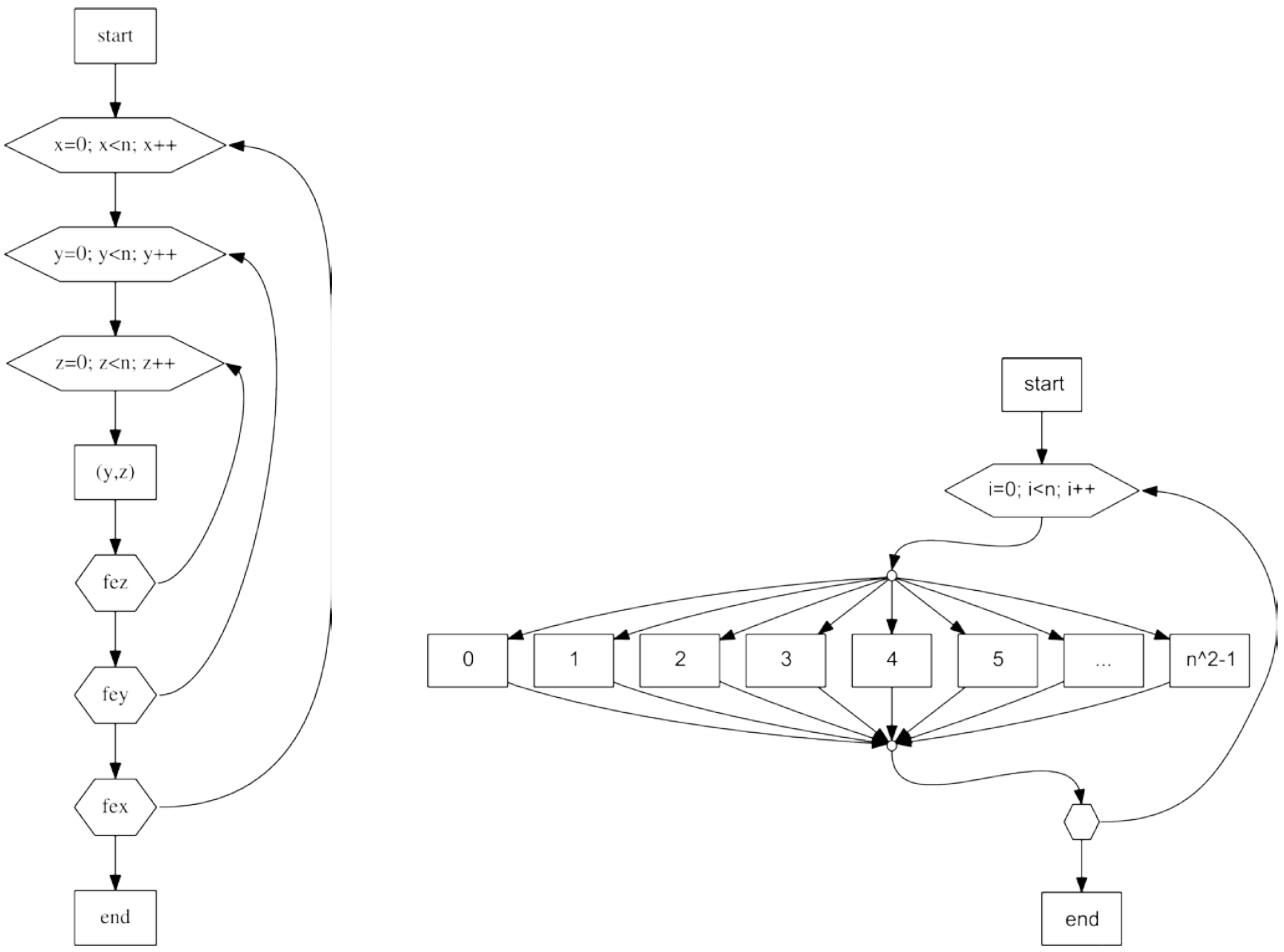

Fig. 2 Flowchart of the Floyd-Warshall algorithm a) *sequential* and b) *parallel*.

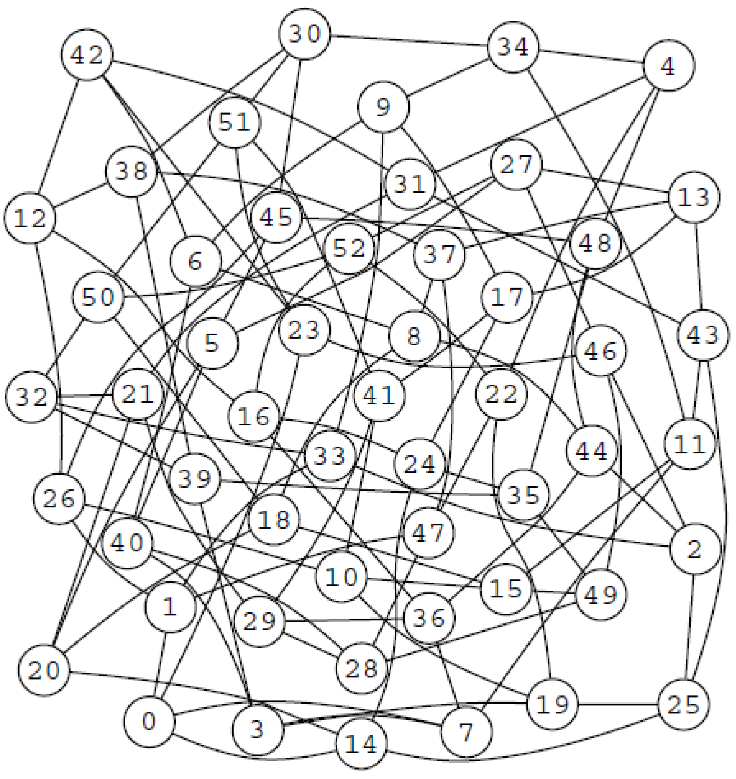

Fig. 3 Final interconnection network for 53 nodes.