HPCLatAm 2013, pp. 65-68 (short paper)
Session: GPU Architecture and Applications
C. Garcia Garino and M. Printista (Eds.)
Mendoza, Argentina, July 29-30, 2013

HPC
High-Performance
Computing - LATAM

Percolation study of samples on 2D lattices using GPUs

D.A. Matoz-Fernandez[1], P.M. Pasinetti[1], and A.J. Ramirez-Pastor[1]

Departamento de Física, Instituto de Física Aplicada, Universidad Nacional de San Luis-CONICET, Ejército de los Andes 950, D5700BWS San Luis, Argentina
fdmatoz@unsl.edu.ar,mpasi@unsl.edu.ar

Abstract. We study the percolation problem of sites on 2D lattices of various geometries, using general purpose graphic processing units (GPGPU). The implementation of a component labeling parallel algorithm in CUDA and their generalization to different geometries, is discussed. The results of performance for this algorithm on a GPU versus the corresponding sequential implementation of reference on a CPU were analyzed. We present different alternatives of implementation, considering the generation of samples in both the CPU host and in the GPU itself, and discussing the synchronization problems that arise. Finally, a new scheme able to take full advantage of the inherent massiveness of the GPU processing simultaneously a great number of samples is presented, showing an significant improvement in the overall performance.

1 Introduction

Since the introduction of graphical processing units (GPUs) for scientific purposes, many problems have been revitalized interest in data parallel computing because they offer a cheap and accessible solution for dedicated data parallel applications. The technological breakthroughs in building graphic processing units of general purpose allowed to win speedups of one or two magnitude orders. The same speedup would be achieved in CPUs after around ten years, respect with the historical 2x performance growth every 2 years of CPU architecture [1].

Certainly one of the most attractive and interesting attributes of GPUs is the high computational throughput due to their large number of processor cores allowing them to perform a huge amount of calculation with a lower cost of time. Notwithstanding, one of the mayor problems of the GPU architectures is the parallel programming paradigm: *Single Instruction Multiple Thread*. In this way the GPU units favoring resolution of problems where the amount of data is very large. This can be seen in the optimal behavior of the distribution of number of threads and blocks required to make full and optimal use of a GPU computing capability. In many scientific applications we don't need to reach large data sizes, thus neglecting all the advantages of the GPUs. In these paper we focus in a better implementation of the percolation problem of sites on 2D lattices using GPUs, from the basic algorithm developed by Kalentev *et.* al. [2], and generalized to various geometries.

The percolation problem has been increasingly considered in statistical physics. One reason for this current interest is that become clear that generalizations of the pure percolation problem are likely to have extensive applications in science and technology. [3,4,5] These applications range from image processing and segmentation to crack

study in material science [6,7]. Although it is a purely geometric phenomenon, the phase transition involved in the process can be described in terms of an usual second-order phase transition. This mapping to critical phenomena made percolation a full part of the theoretical framework of collective phenomena and statistical physics.

The central idea of the pure percolation theory is find the lowest concentration of elements whether or links for which a extends from one side of the system to the opposite side thereof. This particular concentration value is known as the percolation threshold or critical concentration and determines a phase transition in the system [3]. Thus, in the random percolation model, a single site (or link connecting two sites) is occupied with probability p (empty with probability $1 - p$). For the exact value of $p = p_c$, the percolation threshold of sites (links), there is at least one cluster expanded by connecting the edges of the system. In that case, a second order transition appears p_c which is characterized by well-defined critical exponents.

2 Methods

Component Labeling Algorithm

This algorithm is similar to the well-known Hoshen-Kopelman algorithm [8] and their parallel versions [9,2]. It consists in the following steps:

1) Initialization: assign to each occupied site a unique label (*i.e* equal to the site index) and build a list of references.
2) Linking: on each occupied site, choose the smallest nonzero label between $c + 1$ elements, the site itself and its c nearest neighbors. Assign this label to the reference of the site.
3) Relabeling: on each occupied site, search in the references tree until to reach the root, and assign this label to the site.
4) Repeat from step **2**) until there is no more changes in the list of references.

It is worth to note that there is no need of atomic operations to implement the condition in step **4)** which would slowed down the computation dramatically due to their synchronous nature. The search operation in step **3**) works remarkably fast since the steps performed previously make the later ones shorter.

GPU Implementation. We consider samples of $L \times L$ sites arranged on square and triangular geometries. Our implementation is significantly improved by processing simultaneously a great number of samples in such a way to maintain the GPU resources occupation near the optimum (we assign one thread per site). Series of empty border sites (or pads) are used to have open border conditions in each sample, as well as to separate independent samples during the component labeling process (see Fig. 1). This avoid the use of extra conditioning in the border sites. Also, usual optimizations have been applied, such as the use of low-latency memory whenever possible, or the use of optimized CUDA libraries for pseudorandom generation in order to generate the samples in the GPU itself, minimizing the communication with the host through the PCI-e bus. All the Tests were performed on a Intel i7 3770 (3.5GHz), 4GB DD3 (1333), NVIDIA GeForce GTX480 (Fermi, 480 cores), 1.5GB GDDR5, 384-bits, with gcc 4.1.1 (-O3 optimization) and CUDA 4.2.

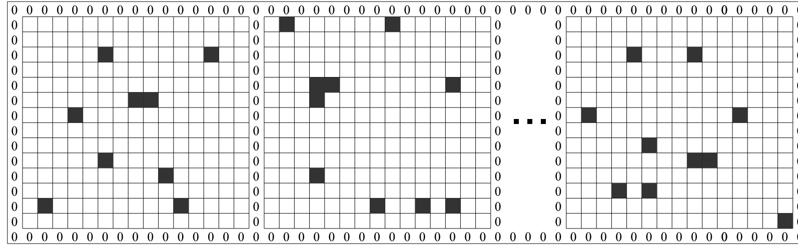

Fig. 1. Simulation scheme of R samples processed simultaneously on GPU. Each sample represents a different disorder realizations. In this way we obtain an optimal use of resources of the GPU. The empty sites in the border (or pads) are represented by zeros in the data matrix and the occupied site is represented by a gray rectangle.

CPU Implementation. The tests were performed on a Intel i7 3770 (3.5GHz), 4GB DD3 (1333). An implementation of the union-and-find algorithm (which is computationally equivalent to the Hoshen-Kopelman algorithm) that was compiled with gcc 4.1.1 (-O3 optimization).

3 Results

For a comparative of the performance, the running time per sample of the serial version of reference on CPU and the corresponding implementation on GPU are shown in Tables 1 and 2. For the sake of clarity, the number of simultaneous samples was fixed to $R = 15$ for all the system sizes so that the resources occupation of the GPU for the largest system was maximum (near 100%). As expected, this methodology is reflected in the size dependence of the speed-up where, for small systems, the utilization of the GPU is far from the optimal, starting with 2% for the first size.

Table 1. Processing time per sample, measured at the critical concentration $p_c = 0.5932$ for the square lattice of $L \times L$ sites.

	Time(ms)			
	256×256	512×512	1024×1024	2048×2048
CPU	10.26	24.15	103.14	530.8
GPUs	8.14	2.04	7.54	26.4
Speed-up	1.26	11.84	13.68	20.11

Table 2. Processing time per sample, measured at the critical concentration $p_c = 0.5$ for the triangular lattice of $L \times L$ sites.

	Time(ms)			
	256×256	512×512	1024×1024	2048×2048
CPU	7.16	25.1	106.3	469.87
GPUs	2.68	5.06	7.15	27.1
Speed-up	2.67	4.96	14.87	17.34

4 Summary

The use of graphic processors for parallel processing shows a significant improvement in performance for the percolation problem. The massively parallel implementation of the component labeling algorithm allows not only detect the occurrence of a percolation cluster but also obtain the complete cluster size distribution of the sample. In general, the study of the percolation systems do not require excessively large sizes to simulate as through finite size scaling techniques we can infer the behavior of the infinite system. To achieve an optimum use of the GPU resources, it is prefered instead to simulate several samples simultaneously, which results in an extra order of magnitude in performance for the largest system size considered. On the other hand, for smaller systems the speed-up reflects the poor occupation of the GPU resources.

Acknowledgments: This work was supported by CONICET (Argentina) under project number PIP 112-200801-01332; Universidad Nacional de San Luis (Argentina) under project 322000; and the National Agency of Scientific and Technological Promotion (Argentina).

References

1. W.W. Hwu, *GPU Computing Gems Emerald Edition* (Elsevier Science, 2011).
2. Oleksandr Kalentev, Abha Rai, Stefan Kemnitz, Ralf Schneider, J. Parallel Distrib. Comput. **71**(4): 615-620 (2011).
3. D. Stauffer, A. Aharony, *Introduction to Percolation Theory*, 2nd ed. (Taylor & Francis, London, 1994).
4. M. Sahimi, *Applications of Percolation Theory* (Taylor & Francis, London, 1994); *Flow and Transport in Porous Media and Fractured Rock* (VCH, Weinheim, Germany, 1995).
5. B. Bollobás, O. Riordan, *Percolation* (Cambridge University Press, New York, 2006).
6. I. Hussain, T.R. Reed, IEEE Transactions on Image Processing **6**(12), 1698-1704 (1997).
7. Tomoyuki Yamaguchi, Shuji Hashimoto, Mach. Vision Appl. **21**(5), 797-809 (2010).
8. J. Hoshen, R. Kopelman, Phys. Rev. B **14**, 3438 (1976); J. Hoshen, R. Kopelman, E. M. Monberg, J. Stat. Phys. **19**, 219 (1978).
9. K.A. Hawick, A. Leist and D.P. Playne, Parallel Computing **36**(12),655-678(2010).