The Review of Design Concepts for the Distributed GPGPU Random Search Computations with the Graphics API Based Core

Oleg Nazarkin
Automated Control Systems Department
Lipetsk State Technical University, Lipetsk, Russia
nazarkino@mail.ru

Abstract

GPGPU methods, which are naturally scalable and natively targeted on the parallel graphics pipeline architecture well-suit for distributed computations. In this paper, the key-points of GPU-oriented distributed processing for the random search tasks is discussed.

Keywords: *GPGPU clusters, GPU-oriented Monte-Carlo implementation, random search.*

1.REASONS FOR GPGPU CLUSTERS

Main reasons are following: 1) cluster give acceleration of data processing to allow interactive or real-time control in the tasks, where it is impossible with a single computer; 2) GPU orientation gives large amount of cheap computational power; 3) naturally scalable, natively targeted on the parallel graphics pipeline architecture, GPGPU methods well-suit for distributed computations with no or little change; 4) more effective load of cluster nodes, as GPU-oriented core utilize CPU power, too.

Random search and evaluation tasks (in a class of Monte-Carlo methods) are among those that best suite for cluster with powerful GPU-equipped computational nodes and narrow-band communications channels (e.g. 100 Mbit/s LAN, or even Internet). Conditional search of multiple-arguments functions extrema is concerned: find $\arg\min_{cond(\vec{X})} F(\vec{X})$ in constraints, defined by $cond(\vec{X})$. Algorithms of function evaluation in every input multidimensional random point, as well as argument constraints checks, if needed, are defined as sequences of GPU instructions (actually they are the parts of program that drives pixel pipeline).

Advantage of graphics APIs (e.g. Direct3D, OpenGL) usage instead of GPGPU APIs (e.g. CUDA, CTM): 1) fitting the non-graphic software into native optimized architectural paradigm of graphics pipeline; 2) independence of specific software, provided by GPU chip manufacturer (no need to setup additional drivers); 3) totally unified application architecture, ease of building homogeneous cluster, from software point of view, on the hardware of different vendors.

The fact that GPUs are still nowadays considered of being hard to program should stimulate the development of effective and easy-to-use frameworks that map computational task elements onto graphics pipeline [1]. The were attempts to utilize graphics engine even before GPUs become flexibly programmable ([2] references the previous implementation of calculations using texture blending).

2.RANDOM SEARCH GPGPU CORE TYPES

The main goal of the core architecture is to provide fast and flexible framework, allowing specification of analytic and table user functions and data. Tasks are divided in 2 groups.

1) Tasks with small search dimensions (up to 8-12 components in $\vec{X}$). For the tasks of this type the framework is able to associate each task variable with a separate texture sampler, thus allowing all data fetches to be parallel and independent. This pattern brings the fastest render pass, due to the most simple data access.

2) Tasks with medium and high search dimensions (up to hundreds components in $\vec{X}$). Instead of separate 2d-textures, bound to individual samplers, multiple layers of single volume texture are used, and all arguments fetches are done by the single sampler. With growth of search function arguments count, the requirements of user data support also grow. The proposed pattern allows more samplers to be bound with user data tables.

The framework implementation should transparently select data layout type, depending on task description.

3.HIERARCHICAL CLUSTER CONSTRUCTION

Implemented computational cores may be wrapped with the components that expose a kind of Remote Object Interface and implement messages exchange through one of the common transport protocols, e.g. TCP/IP. This components represent GPU stations, being the nodes of distributed computational network; special central dispatcher node is needed to control the network. Dispatcher node can implement GPU station interface and, in its turn, perform as a slave GPU-node for higher level dispatchers. This allows to build up hierarchical clusters, useful in that cases, when it is impossible or inefficient to connect available graphics workstations in a "flat" network. Multiple inhomogeneous GPU-nodes may use significantly different cluster control strategies, aiming load-balancing nodes of varying power, or compensation of varying communication segments throughput. The balancing strategies, applied in the intermediate dispatcher nodes, should adjust task solution stages parameters.

4.REFERENCES

[1] D. Goddeke et al. *Exploring weak scalability for FEM calculations on a GPU-enhanced cluster. Parallel Computing 33:10-11. pp. 685-699. 2007.*

[2] Zhe Fan et al. *GPU Cluster for High Performance Computing. Proceedings of the ACM/IEEE SuperComputing 2004 (SC'04), November, 2004*