УДК 004.42
DOI: 10.17277/vestnik.2015.03.pp.406-412

USING PARALLEL BRANCH-AND-BOUND ALGORITHM ON GPUs FOR OPTIMAL DESIGN OF MULTI-PRODUCT BATCH PLANTS

A. B. Borisenko[1], M. Haidl[2], S. Gorlatch[2]

Department "Computer-Integrated Systems in Mechanical Engineering", TSTU (1);
borisenko@mail.gaps.tstu.ru;
Department "Mathematics and Computer Science",
Westfälische Wilhelms-Universität, Münster (Germany) (2)

Keywords: combinatorial optimization; choice of equipment sizes; Compute Unified Device Architecture; multiproduct batch plants; GPU computing; parallel branch-and-bound.

Abstract: In this paper, we propose a novel implementation of the branch-and-bound algorithm on a system with GPUs (Graphics Processing Units) using the Compute Unified Device Architecture (CUDA) approach. The developed implementation is used for solving a real-world problem – the optimal selection of the chemical equipment for multi-product batch plants. We propose two implementations for the searching algorithm – an iterative and a recursive one – and we describe their optimizations, as well as compare them to each other. We report experimental results about the speedup of our GPU-based implementations as compared to the sequential CPU version.

Introduction

Branch-and-bound (**B&B**) is one of the most universal and popular techniques for solving NP-hard combinatorial optimization problems. In the B&B method, the search space can be represented as a tree whose root node is the original unsolved problem, the internal nodes are partially solved subproblems and the leaves are the potential solution(s). B&B proceeds in several iterations during which the best solution found so far (upper bound) is progressively improved. During the exploration, a bounding mechanism, based on a lower bound function, is used to eliminate all the subproblems (i.e., cut their corresponding sub-trees) that are not likely to lead to optimal solutions. Such a powerful mechanism allows reducing significantly the size of the explored search space and, thus, its exploration time cost [1, 2].

In this paper, the focus is on a particular practical application of B&B – the optimal selection of chemical equipment for multi-product batch plants. This kind of plants manufactures a number of related products using the same equipment in (almost) the same operation sequence. There is a great deal of interest in the manufacture of fine chemicals, pharmaceutical products, polymers, and food and beverages using batch operations [3, 4]. Typically, the most common approach used to solve this problem has been to formulate it as a mixed-integer nonlinear programming (**MINLP**) model. It is

well recognized that nonlinear non-convex models are difficult to solve and, with the available solvers, they cannot guarantee global optimal solutions. An option to guarantee global optimality in the solution of the batch design problem is based on the development of linear models. Many non-linear models for batch design, which are based on the assumption of continuous sizes, can be reformulated as MILP problems when sizes are restricted to discrete values [5, 6]. We reduce the computational effort for our application by using the B&B approach which is a popular technique for solving optimization problems in various fields (e.g., combinatorial optimization, artificial intelligence, etc.), including MINLPs and MILP [7, 8]. In this paper, we develop and evaluate an implementation of the B&B method on a CPU-GPU system using the CUDA programming environment [9]. We report experimental results about the speedup of our GPU-based implementations as compared to the sequential CPU version.

Problem Formulation

A chemical-engineering system (**CES**) is a set of equipment (reactors, tanks, filters, dryers etc.) which implement the processing stages for manufacturing certain products. Assuming that the number of units at every stage of CES is fixed, the problem can be formulated as follows: CES consists of a sequence of I processing stages. Each *i*-th processing stage of the system can be equipped with equipment units from a finite set X_i, with J_i being the number of equipment units variants in X_i. All equipment unit variants of a CES are described as $X_i = \left\{x_{i,j}\right\},\ i = \overline{1,I},\ j = \overline{1,J_i}$, where $x_{i,j}$ is the main size *j* (working volume, working surface) of the unit suitable for processing stage *i*.

A CES variant $\Omega_e,\ e = \overline{1,E}$ of a CES, where $E = \Pi_{i=1}^{I} J_i$ is the number of all possible system variants, is an ordered set of equipment unit variants, selected from the respective sets.

Each variant Ω_e, of a system must be in operable condition (*compatibility constraint*) i.e., it must satisfy the conditions of a joint action for all its processing stages: $S(\Omega_e) = 0$ if compatibility constraint is satisfied. An operable variant of a CES must run at a given production rate in a given period of time (*processing time constraint*), such that it satisfies the restrictions for the duration of its operating period $T(\Omega_e) \le T_{\max}$, where $T_{\max}$ is a given maximum period of time.

Thus, designing an optimal CES can be formulated as the following optimization problem: to find a variant $\Omega_e^* \in \Omega_e,\ e = \overline{1,E}$ of a CES, where the optimality criterion – equipment costs $\mathrm{Cost}(\Omega_e^*)$ reaches a minimum and both compatibility constraint and processing time constraint are satisfied:

$$\Omega_e^* = \operatorname{argmin} \mathrm{Cost}(\Omega_e),\ e = \overline{1,E};$$

$$\Omega_e = \left\{x_{1,j_1}, x_{2,j_2}, ..., x_{I,j_I} \mid j_i = \overline{1,J_i},\ i = \overline{1,I}\right\},\ e = \overline{1,E};$$

$$x_{i,j} \in X_i,\ i = \overline{1,I},\ j = \overline{1,J_i};$$

$$S(\Omega_e) = 0,\ e = \overline{1,E};$$

$$T(\Omega_e) \le T_{\max},\ e = \overline{1,E}.$$

We use the comprehensive mathematical model of CES operation, including expressions for checking constraints, calculating the optimization criterion, etc., which was initially presented in [10].

Parallelization for GPU

Modern computers increasingly become heterogeneous, with different types of computational units. For example, multicore systems gain performance not only by adding cores, but also by incorporating specialized processing capabilities (Graphics Processing Units (**GPUs**), Digital Signal Processors (**DSPs**), etc.) to handle particular tasks. GPUs are currently the most powerful computational hardware available at an affordable price. Assisting the Central Processing Unit (**CPU**), GPUs nowadays not only render 3D graphics, but also perform high-performance, general-purpose computations [11].

The GPU-accelerated approach is based on the programming paradigm according to which the programmer writes a serial, so-called host program that runs on the CPU and calls parallel kernels (simple functions or full programs) on one or several GPUs. In this paper, we investigate an implementation of the B&B method on a CPU-GPU system via CUDA [9].

To facilitate a structured exploration of the search space, it is considered as a tree: all possible variants of a CES are represented by a search tree of height *I*. Each level of the tree corresponds to one processing stage of the CES, each edge corresponds to a selected equipment variant taken from set X_i, where X_i is the set of possible variants at stage i of the CES. For example, the edges from level 0 of the tree correspond to elements of X_1. Each node $n_{i,k}$ at the tree layer $N_i = \{n_{i,1}, n_{i,2}, \ldots, n_{i,k}\}$ $i = \overline{1, I}$, $k = \overline{1, K_i}$, $K_i = \Pi_{l=1}^{i} J_l$ corresponds to a variant of a beginning part of the CES, composed of equipment units for stages 1 to *i* of the CES. Each path from the tree's root to one of its leaves thus represents a complete variant of the CES. To enumerate all possible variants of a CES in the tree, a depth-first traversal of the tree is performed as shown in Figure 1: starting at level 0, all device variants of the CES at a given level are enumerated and appended to the valid beginning parts of the CES. Valid beginning parts are obtained at previous levels, starting with an empty beginning part at level 0. This process continues recursively for all valid beginning parts that result from appending device variants of the current level to the valid beginning parts from previous levels. When a leaf is reached, the recursive process stops and the current solution is compared to the current optimal solution, possibly replacing it.

To divide the initial search tree into subtrees for parallels processing, we use the static strategy illustrated in Fig. 1. A sequential *host* process on the CPU dispatches a

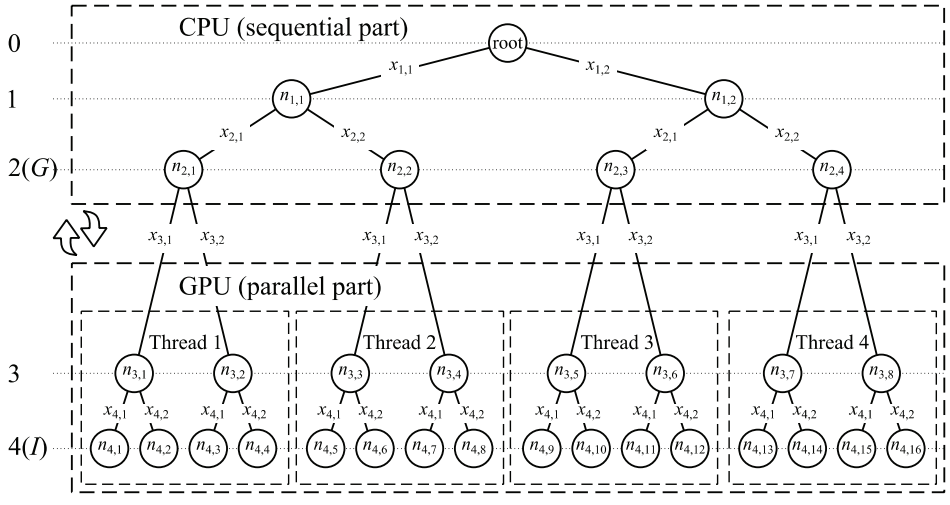

Fig. 1. Dividing the search tree into subtrees for parallel processing

subset of computations to multiple *device* threads on the GPU and then gathers computed results from these threads.

For the tree traversal we used both recursive and iterative approaches. Using stack (sometimes also called Last-In-First-Out (**LIFO**) data structure) is the obvious way to traverse tree without recursion. CUDA encompasses two application programming interfaces: the CUDA Driver API and the CUDA Runtime API. The low-level driver API provides explicit control over GPUs. It is implemented by a runtime library and can be used with any standard C compiler. The runtime API is based on an additional runtime library and the CUDA C programming language. To develop a parallel program on the GPU, we use the C programming language with the CUDA Runtime API. Unlike the driver API, the runtime API does not require explicit initialization in the host program in order to use a GPU: all necessary initialization steps are performed automatically when the first function of the runtime API is called. The recursive approach can be used on NVIDIA GPU devices of Compute Capability 2.0 and higher. On older NVIDIA devices, the iterative approach should be used. The NVIDIA GPU device have some Streaming Multiprocessors (**SMs**) contains some Streaming Processors (**SP**) (since Fermi microarchitecture NVIDIA changes the name SP to *CUDA cores*). These SMs only get one instruction at a time which means that the CUDA cores all execute the same instruction. Because threads (and not data) are mapped to the CUDA cores and executed in the *Si*ngle-Instruction, Multiple-Data (**SIMD**) like fashion, the style of execution is called Single-Instruction, Multiple-Thread (**SIMT**): each thread executes the same instruction, but possibly on different data. Threads within a *warp* (groups of 32 threads, which use in the hardware implementation to coalesce memory access and instruction dispatch) must execute the same instruction each cycle. When the execution encounters a divergent control flow, the SM is forced to execute both control-flow paths to the point where they join back together. The warp serially executes each control-flow path taken, disabling threads that are not on that path (also known as *branch divergence*). We examine two optimizations that aim at improving the performance of our programs: shared memory (the fastest on device, but only 48 KB on each SM), as well as utilizing and reducing brunching in the kernel function.

Experimental Results

Our experiments were conducted on a system comprising: an Intel Xeon processor (E5-1620 v2, 4 cores with Hyper-Threading Technology, Smart Cache 10MB, running at 3.7 GHz) with 16 GB RAM; a GPU NVIDIA Tesla K20c which features 13 SM with 192 CUDA Cores (total 2496 CUDA Cores), 5GB of global memory and up to 48KB of shared memory per SM. We use Ubuntu 14.04.1, NVIDIA Driver version 340.29, CUDA version 6.5 and GNU C++ Compiler version 4.8.2.

We study the design of a CES consisting of 16 processing stages with 3 variants of devices at every stage as test case (total 3^{16} = 43 046 721 CES variants), with 4 variants of devices at every stage (total 4^{16} = 4 294 967 296 CES variants) and with 5 variants of devices at every stage (total 5^{16} = 152 587 890 625 CES variants).

In Figure 2, the speedups as compared to the sequential CPU-program for both tree traversal implementations are presented.

We observe in Fig. 2 that the speedup value for the recursive version is between 2.66 and 5.79, and for the iterative version it is between 1.67 and 4.43. In all the presented test examples, the recursion is faster than the iteration. Based on these experiments, we can conclude than the direct implementation of the B&B algorithm on GPUs did not produce any significant speedup. The most obvious problem with tree traversal on GPU is a high warp divergence due to many control-flow execution paths (such as *if-else* instructions). Each tree node processed in a parallel manner represents a different CES variant, therefore every searched subtree may have a different structure. Similar problems have been described, for example, in the article [12] which describes

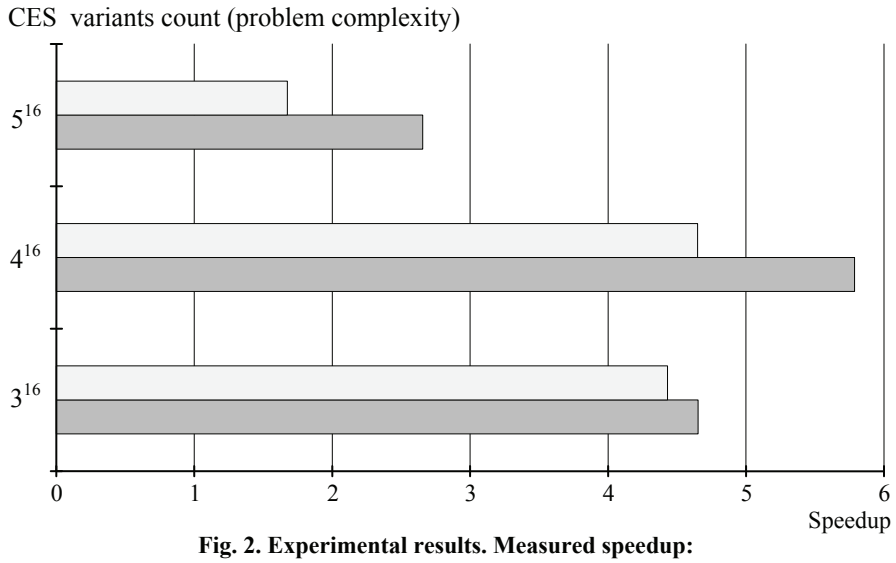

Fig. 2. Experimental results. Measured speedup:
□ – Iteration; ■ – Recursion

the results of a minimax tree searching algorithm, also implemented using the CUDA approach.

Our future work can have several directions. First, we can try to use multiple GPUs. Some graphics units can work independently and algorithm would scale well enough. Second, we can optimize our implementation of the mathematical model of CES for working within GPUs hardware restrictions (reducing divergence, improving memory utilization etc.). Finally, we plan to use CPU and GPU simultaneously for tree traversing. In this case, we will need to develop another parallel algorithm and corresponding load-balancing methods.

References

1. Melab, N., Chakroun I., Mezmaz M., Tuyttens D. *2012 IEEE International Conference on Cluster Computing*, Beijing, China, 24 – 28 Sept. 2012, pp. 10-17.

2. Borisenko A. B., Kutuzov D. V., Osovskii A. V. *Transactions of the Tambov State Technical University*, 2011, vol. 17, no. 2, pp. 493-496.

3. Seid E. R. Majozi T. *Ind. Eng. Chem. Res.*, 2013, vol. 52, no. 46, pp. 16301-16313.

4. Borisenko A. B., Antonenko A. V., Osovsky A. V., Filimonova O. A. *Transactions of the Tambov State Technical University*, 2012, vol. 18, no. 3, pp. 569-572.

5. Fumero Y., Corsano G., Montagna J. M. *Appl. Math. Model.*, 2013, vol. 37, no. 4, pp. 1652-1664.

6. Moreno M. S., Montagna J. M. *AIChE J.*, 2011, vol. 57, no. 1, pp. 122-135.

7. Burer S., Letchford A. N. *Surv. Oper. Res. Manag. Sci.*, 2012, vol. 17, no 2, pp. 97-106.

8. Borisenko A., Kegel P., Gorlatch S. *Lecture Notes in Computer Science*, 2011, vol. 6873, pp. 417-430.

9. NVIDIA Corporation, *CUDA C Programming Guide 6.5*, 2014.

10. Malygin E. N., Karpushkin S. V., Borisenko A. B. *Theor. Found. Chem. Eng.*, 2005, vol. 39, no. 4, pp. 429-439.

11. Sluga D., Curk T., Zupan B., Lotric U. *BMC Bioinformatics*, 2014, vol. 15, no. 1, p. 216.

12. Rocki K., Suda R. *Lecture Notes in Computer Science*, 2010, vol. 6067, pp. 449-456.

Применение параллельного алгоритма ветвей и границ на графических процессорах для оптимального проектирования многоассортиментных производств

А. Б. Борисенко[1], М. Хайдл[2], С. Горлач[2]

Кафедра "Компьютерно-интегрированные системы в машиностроении", ФГБОУ ВПО «ТГТУ» (1); borisenko@mail.gaps.tstu.ru; кафедра "Математика и информатика", Вестфальский университет имени Вильгельма, г. Мюнстер, Германия (2)

Ключевые слова: выбор оборудования; вычисления на GPU; комбинаторная оптимизация; метод ветвей и границ; многоассортиментные производства; параллельные вычисления.

Аннотация: Представлена параллельная реализация алгоритма ветвей и границ для вычислительных систем, оснащенных графическими процессорами GPU (Graphics Processings Units) с использованием технологии CUDA (Compute Unified Device Architecture). Алгоритм используется для решения задачи оптимального выбора аппаратурного оформления многоассортиментных производств. Приведены две программных реализации обхода дерева поиска – итеративная и рекурсивная. Дано описание их оптимизации и представлен сравнительный анализ быстродействия. Показаны результаты вычислительных экспериментов для тестовых задач различной размерности: значения ускорений параллельной программы с использованием GPU относительно последовательного варианта программы.

Список литературы

1. A GPU-Accelerated Branch-and-Bound Algorithm for the Flow-Shop Scheduling Problem / N. Melab [et al.] // 2012 IEEE International Conference on Cluster Computing, Beijing, China, 24 – 28 Sept. 2012. – P. 10 – 17.

2. Борисенко, А. Б. Применение параллельных вычислений для расчета аппаратурного оформления химико-технологических систем / А. Б. Борисенко, Д. В. Кутузов, А. В. Осовский // Вестн. Тамб. гос. техн. ун-та. – 2011. – Vol. 17, No. 2. – С. 493 – 496.

3. Seid, E. R. Design and Synthesis of Multipurpose Batch Plants Using a Robust Sscheduling Platform / E. R. Seid, T. Majozi // Ind. Eng. Chem. Res. – 2013. – Vol. 52, No. 46. – P. 16301 – 16313.

4. Система автоматизированного выбора вспомогательного оборудования многоассортиментных химических производств / А.Б. Борисенко [и др.] // Вестн. Тамб. гос. техн. ун-та. – 2012. – Т. 18, № 3. – С. 569 – 572.

5. Fumero, Y. A Mixed Integer Linear Programming Model for Simultaneous Design and Scheduling of Flowshop Plants / Y. Fumero, G. Corsano, J. M. Montagna // Appl. Math. Model. – 2013. – Vol. 37, No. 4. – P. 1652 – 1664.

6. Moreno, M. S. Multiproduct Batch Plants Design Using Linear Process Performance Models / M. S. Moreno, J. M. Montagna // AIChE J. – 2011. – Vol. 57, No. 1. – P. 122 – 135.

7. Burer, S. Non-Convex Mixed-Integer Nonlinear Programming: A survey / S. Burer, A. N. Letchford // Surv. Oper. Res. Manag. Sci. – 2012. – Vol. 17, No. 2. – P. 97 – 106.

8. Borisenko, A. Optimal Design of Multi-Product Batch Plants Using a Parallel Branch-And-Bound Method / A. Borisenko, P. Kegel, S. Gorlatch // Lecture Notes in Computer Science. – 2011. – Vol. 6873. – P. 417 – 430.

9. NVIDIA Corporation. CUDA C Programming Guide 6.5. – 2014.
10. Malygin, E. N. A Mathematical Model of the Functioning of Multiproduct Chemical Engineering Systems / E. N. Malygin, S. V. Karpushkin, A. B. Borisenko // Theor. Found. Chem. Eng. – 2005. – Vol. 39, No. 4. – P. 429 – 439.
11. Heterogeneous Computing Architecture for Fast Detection of SNP-SNP Interactions / D. Sluga [et al.] // BMC Bioinformatics. – 2014. – Vol. 15, No. 1. – P. 216.
12. Rocki, K. Parallel Minimax Tree Searching on GPU / K. Rocki, R. Suda // Lecture Notes in Computer Science. – 2010. – Vol. 6067. – P. 449 – 456.

Anwendung des parallelen Algorithmus der Äste und der Grenzen auf den graphischen Prozessoren für die optimalen Projektierung der vielsortimenten Produktionen

Zusammenfassung: Es ist die parallele Realisierung des Algorithmus der Äste und der Grenzen für die Rechensysteme, die mit den graphischen Prozessoren (Graphics Processings Units – GPU) mit der Benutzung der Technologie Compute Unified Device Architecture (CUDA) ausgestattet sind, dargelegt. Der Algorithmus wird für die Lösung der Aufgabe der Optimalen Auswahl der Apparaturgestaltung der vielsortimenten Produktionen benutzt. Es werden zwei Programmrealisierungen der Umgehung des Suchebaumes – iterative und recursive angeführt, es wird ihre Optimierung beschrieben und es wird die Vergleichanalyse der Schnellwirkung angeführt. Es sind die Ergebnisse der Rechenexperimente für die Testaufgaben der verschiedenen Dimension dargelegt: die Werte der Beschleunigungen des Parallelprogrammes mit dem Benutzung von GPU bezüglich des konsequenten Variantes des Programms.

Application d'un algorithme parallèle des branches et des frontières sur les processeurs graphiques pour la conception optimale des productions multi-gammes

Résumé: Est présentée la réalisation parallèle de l'algorithme de branches et les limites pour l'informatique dans les systèmes équipés de processeurs graphiques (Graphics Processings Units – GPU) avec l'aide de la technologie Compute Unified Device Architecture (CUDA). L'algorithme est utilisé pour résoudre le problème du choix optimal des algorithmes des productions multi-gammes. Sont cités deux logiciels de la mise en œuvre de parcours de l'arbre de recherche – itératif et récursif; est décrite leur optimisation; est effectuée une analyse comparative des performances. Sont présentés les résultats de calcul d'expériences pour les problèmes de test à différentes dimensions: les valeurs du problème parallèle avec l'emploi de GPU relativement la variante séquentielle du programme.

Авторы: ***Борисенко Андрей Борисович*** – кандидат технических наук, доцент кафедры «Компьютерно-интегрированные системы в машиностроении», ФГБОУ ВПО «ТГТУ»; ***Хайдл Михаэль*** – аспирант кафедры «Математика и информатика»; ***Горлач Сергей*** – PhD, профессор кафедры «Математика и информатика», Вестфальский университет имени Вильгельма, г. Мюнстер, Германия.

Рецензент: ***Гатапова Наталья Цибиковна*** – доктор технических наук, профессор, заведующая кафедрой «Технологические процессы, аппараты и техносферная безопасность», ФГБОУ ВПО «ТГТУ».