Clustering-Based Algorithm for Workload Allocation to Heterogeneous Processors with Constraint on Interprocessor Data Exchange*

Vasily V. Balashov, Yaroslav A. Basalov

Abstract — **This paper addresses the problem of allocating the workload (task graph) to a set of processors with different performance, under constraint on the number of interprocessor data transfers. Such problems arise during development of real-time data processing systems, for example telecommunication and control systems. This problem is a generalization of the weighted graph partitioning problem with constraint on the number of cut-edges. In contrast to the original problem, the node weight (task execution time) depends on the node group number (i.e. processor). An algorithm is proposed, which uses the METIS partitioner to contain the "extra" edges in multiple subgraphs (clusters) and guarantee meeting the constraint, then proceeds with the greedy phase to distribute these subgraphs between processors. Experimental evaluation results are presented, demonstrating the advantage of the proposed algorithm over the greedy algorithm without preliminary clustering.**

I. Introduction

The problem considered in this paper arises in the context of multiprocessor scheduling for a system with heterogeneous (having different performance) fully connected processors. The workload is specified as a directed acyclic task graph (DAG) with nodes corresponding to tasks, and edges to task dependencies as well as inter-task data transfers. Node weight is the basic duration of a task. For every processor, its performance is specified as a positive number referred to as speed factor. To obtain task duration for a specific processor, the task's basic duration is divided by the processor speed factor.

A constraint is imposed on the share of interprocessor data transfers: it must not exceed the specified bound. As noted in [1], such constraint is relevant if the interprocessor network is not exclusively reserved for data exchange between tasks of the given DAG, and there is a need to retain some network resource for other purposes, which have lower priority and thus introduce no direct interference. This corresponds to a mixed-criticality system, with the given task graph being the most critical workload. Taking in account only the number of data transfers, but not the message sizes, is reasonable when all messages have approximately the same size. This is typical e.g. for voice processing, where message size is predefined.

* V. V. Balashov is with Lomonosov Moscow State University, Department of Computational Mathematics and Cybernetics, Leninskie Gory, MSU, 1, Bldg. 52, Room 764, Moscow, Russia (corresponding author, phone: +7(495)939-4671; fax: +7(495)939-2596; e-mail: hbd@cs.msu.ru).

Y. A. Basalov is with Lomonosov MSU, CMC Department (e-mail: BasalovYA@my.msu.ru).

To meet this constraint on the schedule, tasks allocation to processors should be done before schedule construction. If the tasks allocation is made in course of scheduling, the algorithm may come to a dead end, where placement of a task to any processor will violate the constraint.

The problem can be formalized as a generalization of the k-way weighted graph partitioning problem with constraint on the share of cut-edges (cut-ratio, or CR), where k is the number of processors. Partitioning of the graph produces node groups (subgraphs, or parts). Every group of nodes (tasks) is the workload for a specific processor. Interprocessor data transfers correspond to cut-edges of the partition, i.e. edges between nodes of different groups.

In case of processors with same performance, the partition must be balanced in order to level the load on the processors. The problem of finding a balanced (as far as possible) partition while meeting the CR constraint is well explored and can be solved with existing tools, e.g. METIS [2].

However, if the processors have different performance, balancing the load may be harmful for the schedule. For processors with speed factors of 1 and 10, and nine independent tasks with basic duration 100, it is better to place all tasks on the fast processor (schedule duration: 90) than to move one task to the slow processor for load balancing (schedule duration: 100). Therefore, a different optimized criterion is needed: maximum, by all processors, total duration of tasks on this processor. This criterion estimates the schedule duration from below, and must be minimized.

Introduction of processor heterogeneity via speed factors, along with CR constraint, leads to the task graph partitioning problem, which is not solved by available algorithms. In this paper, we propose an algorithm for solving this problem.

II. Problem Statement

The problem of tasks allocation to heterogeneous processors is stated as a task graph partitioning problem. First, we introduce the necessary definitions.

Task graph (application program model) is specified as a DAG $G=(V, E)$ with $|V|$ nodes and $|E|$ edges. Every node of G corresponds to a task from the task set $\{v_i\}_{i=0}^{|V|-1}$. Basic duration of the task v_i is t_i. Every edge in G corresponds to a data transfer between tasks.

Computer system model. The computer system M is a set of k fully connected processors with different performance:

$M = \{m_j\}_{j=0}^{k-1}$. Every processor m_j is characterized by its speed factor q_j. Duration (execution time) of the task v_i on the processor m_j is calculated as: $t_i^j = t_i / q_j$. Every task must be allocated to a single processor.

Task graph partition $P = \{P_j\}_{j=0}^{k-1}$ is defined for a given task graph G and set of processors M as a mapping $V \rightarrow M$. This mapping specifies the tasks allocation to processors. P_j is the set of tasks allocated to the processor m_j, or the j-th *part* of the partition.

Cut-ratio of the partition: $CR=N_P/|E|$, where N_P is the number of edges between nodes from sets P_j with different j, i.e. the number of data transfers between tasks allocated to different processors.

Partition P is *correct*, if it meets following constraints:

1) Every task is assigned to a single processor;
2) CR constraint is met: $CR \le CR_U$, where CR_U is a specified upper bound.

The fact that partition P meets constraints 1 and 2 is denoted as $P \in P_{1,2}^*$.

Total duration of tasks allocated to the processor m_j:

$$T_j = \sum_{v_i \in P_j} t_i^j$$

Objective function: partition execution time

$$f(P) = \max_{j=0..k-1} T_j$$

Problem statement:

Given: task graph G, set of processors M, upper bound on cut-ratio CR_U.

Required: construct a partition P of the task graph.

Minimized criterion: $f(P)$.

Constraints: $P \in P_{1,2}^*$.

Let us abbreviate this problem as HPCR, for "Heterogeneous Partitioning with Cut-Ratio constraint".

We choose the partition execution time $f(P)$ as the minimized criterion, because the total duration of tasks on a specific processor provides a lower estimate for schedule duration on this processor, therefore $f(P)$ estimates the duration of the whole schedule from below. Precision of such estimate depends on the structure of G. For instance if most of the tasks are grouped into several long chains, the number of processors is by many times greater than the number of these chains, and each chain is distributed between multiple processors, then $f(P)$ significantly underestimates the schedule duration because tasks in every chain must be executed sequentially. With more task chains and less processors, the precision of $f(P)$ as a lower estimate for schedule duration will be better. Taking in account the structure of paths in G, as well as construction of the schedule itself, is beyond the scope of algorithms proposed in this paper.

The HPCR problem is NP-hard, because its particular case is Problem SP12 ("Partition") in [3]. In such case, CR_U is 1.0 (no constraint), G has no edges (the tasks are independent), and M contains two identical processors.

Farther in this paper, we consider the terms "task" and "node" interchangeable.

III. Related work

For the singular case of identical processors, minimization of partition execution time $f(P)$ is closely related to finding a balanced partition. For instance, the partition with zero imbalance (all processors have same total duration of tasks) also minimizes $f(P)$.

For identical processors, the problem of finding a partition with minimum imbalance while meeting the CR constraint can be solved with existing algorithms and tools, e.g. METIS [2]. METIS minimizes the cut-ratio while meeting the constraint on imbalance (min-cut problem), but this tool can be used iteratively with gradual increase of imbalance limit, until cut-ratio becomes less or equal to CR_U.

Adding processor heterogeneity (while keeping the CR constraint) significantly narrows down the list of existing approaches to solving HPCR. Several large surveys of graph partitioning algorithms [4 – 8], with over 500 references in total, were analyzed, resulting in some workload allocation algorithms taking in account processor heterogeneity, and very few of them considering CR-like constraints. By "CR-like", we mean constraints that can be used to simulate the original CR constraint. An example is the constraint on the total size of messages passed between processors: by setting the size of every message to 1, we make the total size of messages equal to the number of cut-edges in the partition.

From now on, we focus on workload (task graph) allocation algorithms, as the weighted graph partitioning algorithms not related to workload allocation typically keep node weights independent of the node group (part) in a partition.

Thesis [8] is focused on architecture-aware graph partitioning, and identifies only two algorithms [9, 10] as supporting the processor heterogeneity. The algorithm from [9] does not support CR-like constraints. Moreover, it is a streaming algorithm, allocating tasks to nodes in the course of tasks arrival. Such scheme is hardly compatible with CR constraint, because sequential (one-by-one) assignment of the nodes to processors may lead to a dead end, when the next node cannot be assigned to any processor without violating the CR constraint. The paper [10] by the author of METIS proposes to apply this tool for initial partitioning of the task graph (meeting the CR constraint), and to use a greedy algorithm to refine the partition taking in account the processor heterogeneity. The refinement is performed by moving the tasks between parts (i.e. processors) as long as such moves decrease the partition execution time. We take this approach as a baseline; however, our experiments show that partition refinement phase often stops on a partition with

mediocre quality, because moving any task from the most loaded processor violates the CR constraint.

In the work [8] itself, only network heterogeneity-aware algorithms are proposed. Processor heterogeneity is not taken in account. A major part of architecture-aware partitioning algorithms for task graphs is oriented at network heterogeneity only.

More streaming workload allocation algorithms for heterogeneous processors are proposed in [11], with same obstacles to incorporation of CR constraint as in [9]. As these obstacles are applicable to streaming partitioning algorithms in general, we will consider below only those algorithms that take the whole task graph as input.

GraphIVE tool [12] supports processor heterogeneity and does not support CR-like constraints. This tool focuses on rebalancing the workload while learning the performance of processors in course of tasks execution. Advantages of GraphIVE are in its task migration capabilities, which are not relevant to the HPCR problem.

The GAP approach proposed in [13] supports processor heterogeneity, minimizes partition execution time, and does not support CR-like constraints. Partitioning is performed in two stages: first, the tasks are allocated to the processors in quantities proportional to processor speed factors; second, the partition is refined by a genetic algorithm that moves tasks between processors. For a dense graph, the first stage produces a partition very far from meeting the CR constraint, and the second-stage genetic algorithm is not aimed at decreasing the cut-ratio.

In [14] the algorithm minimizes the partition execution time on heterogeneous processors, without support for CR-like constraints. This algorithm has three phases: downscaling (clustering) of the task graph by collapsing the highly connected subgraphs (clusters of nodes) into nodes of the compressed graph; partitioning of the compressed graph (i.e. allocating its nodes to processors); decompressing it and refining the resulting partition. Such approach, with several clustering steps, is also used in METIS [2]. The task graph clustering step will be used in the algorithm proposed in the present paper to guarantee meeting the CR constraint regardless of clusters allocation to processors, and to avoid the CR-related dead-end problem.

In [15] the partitioning algorithm maximizes the throughput of a stream processing application represented as a task graph (not to be confused with streaming workload allocation). It is based on Kernighan-Lin algorithm [16] adapted to k-way partitioning, and moves nodes between parts (processors) one by one. This scheme is prone to dead-ends in presence of CR constraint, especially if nodes are grouped in highly connected subsets.

The genetic algorithm in [17] minimizes the total weight of cut-edges in the partition under per-processor limits on processor load (total duration of tasks). Heterogeneity of the processors can be simulated by setting different limits on processor load. These limits can be gradually increased until the CR constraint is met, but having a genetic algorithm in the enumeration loop is hardly acceptable due to scalability reasons.

Conclusions from the overview are: (1) no algorithms for solving HPCR problem are readily available; (2) existing algorithms for workload allocation to heterogeneous processors with minimization of partition execution time are ill suited for incorporation of CR constraint; (3) several ideas from existing algorithms can be used for solving HPCR (see the description of proposed algorithms for details).

IV. Proposed partitioning algorithms

A. Basic greedy algorithm

The greedy algorithm is inspired by the two-stage scheme from [10], in which the initial partition is constructed by METIS and then refined by moving tasks between parts (processors) while such moves improve the objective function. Note that [10] contains only general description of this scheme.

The greedy algorithm (denoted A_{gr}) consists of following steps:

1) Using METIS, construct a weighted partition of task graph G into k subgraphs, meeting the CR constraint and as balanced as possible under this constraint;

Note: k is the number of processors, but for strict CR constraint (small CR_U) and/or dense task graph, the number of subgraphs in the partition can be less than k.

2) Assign every subgraph to a separate processor; for subgraphs with greater total weight of nodes (basic durations of tasks), choose processors with greater speed factor;

3) Choose the most loaded processor m_i (with greatest value of T_i);

4) Consider the tasks assigned to m_i in order of basic task duration decrease. For the current task v:

   a) choose the processor m_j with greatest speed factor from processors meeting following condition: after moving v from m_i to m_j, the value of objective function $f(P)$ decreases, and the CR constraint remains met;

Note: moving a task from a more loaded processor a to a less loaded processor b leads to increase of $f(P)$ if the total duration of tasks on b after such move is greater than the total duration of tasks on a before the move.

   b) if the processor m_j meeting the condition is successfully chosen in step 4.a, then: move the task v from m_i to m_j; if m_i is no longer the most loaded processor, go to step 3;

   c) if there are unconsidered tasks on m_i, continue with step 4, else **stop**.

Preliminary experiments indicated that for strict CR constraint and/or dense task graph the greedy algorithm often stops in step 4.a because no single task can be moved from the most loaded processor without violating the constraint. In many such dead-end cases, a locally connected group of tasks could be moved from the processor; this suggests the idea of using such groups as the units for initial workload allocation and its refinement. The algorithm in the next subsection is based on this idea.

B. Clustering-based algorithm

This algorithm begins with clustering the task graph G by collapsing the highly connected subgraphs (clusters of nodes) into nodes of a new undirected graph G^*, referred to as the cluster graph. A node in G^* has weight equal to the total weight of nodes in G which were collapsed into it. There is a single undirected edge in G^* between the nodes v and w if there is at least one edge in G between some node collapsed into v and some node collapsed into w.

Transition to the cluster graph is a common step in partitioning algorithms. For instance, in [2, 14] it is used to scale down the graph and proceed with coarse-grain partitioning. In our clustering-based algorithm, such transition is performed to get as many clusters as possible while keeping the share of cut-edges between the clusters under CR_U. Thus we get a cluster graph G^* in which the CR constraint in its original meaning (share of edges between tasks assigned to different processors) will be met regardless of assignment of whole clusters to processors. CR constraint will be met even if each cluster is assigned to a separate processor, and by assigning several clusters to the same processor, we exclude the edges between nodes of these clusters from the set of inter-processor edges, decreasing the share of inter-processor edges. So the benefit of working with nodes of G^* is that during their allocation to processors there is no need to care for CR constraint, as it is already met by construction of G^*. "Absence" of CR constraint during allocation of G^* nodes to processors removes the main drawback of the greedy algorithm A_{gr} described in the previous subsection, namely its tendency to get into dead-end due to this constraint during partition refinement.

Finding G^* with as many nodes as possible provides flexibility in combining these nodes on processors. However, the preliminary experiments indicated that partitioning of task graphs to the maximum number of subgraphs under CR constraint results in several very large subgraphs (containing most of the edges) and multiple small subgraphs, forcing some processors to get those large subgraphs and be overloaded. So we decided to perform enumeration on the target number of subgraphs (nodes of G^*), as both METIS and the greedy algorithm work fast enough.

The clustering-based algorithm (denoted A_{cl}) has the following scheme:

1) Find the maximum number of parts L_{max}, to which the task graph G can be partitioned under CR constraint. To do this, perform binary search on the interval 1,..,$|V|$. For every value of l checked during this search, use METIS for weighted partitioning of G to l parts under CR constraint, starting with strict imbalance limit (*ufactor*=1) and relaxing it (doubling *ufactor*) until a partition meeting the CR constraint is constructed or the specified upper bound on *ufactor* is reached.

   Note: METIS sometimes constructs partitions with less than requested number of parts. Such partitions are discarded.

2) For each j in 1,…, L_{max}:

   a) Using METIS, find a weighted partition of G to j parts under CR constraint, with as little imbalance as possible. To do this, find some value of *ufactor* with which the partition is successfully constructed (by doubling *ufactor* as in step 1) and try to find a lesser value of *ufactor* with binary search.

   b) If for current j the partition is not found in step 2.a, continue with next j;

   c) Construct the set of tasks W with basic durations equal to total basic durations of tasks in the parts of the resulting partition of step 2.a;

   Note: W is the set of nodes in the cluster graph G^*. The edges of G^* are not processed by this algorithm.

   d) Using the greedy algorithm A_{gr}, allocate the tasks from W to processors. CR constraint is not checked in this step, as it is met by construction in step 2.a;

   e) "Unpack" the tasks of W to get the partition of the original task graph G's nodes (i.e. tasks allocation to processors);

   f) use A_{gr} with initial partition from step 2.e to locally improve this partition, and continue with next j;

3) Choose the best partition P (with minimum execution time $f(P)$) of those constructed in step 2;

4) **Stop**.

This algorithm combines the greedy scheme of A_{gr} with limited enumeration by the number of parts in the partition and the value of imbalance limit (*ufactor*).

Complexity of A_{cl} algorithm is dominated by multiple runs of METIS in steps 1 and 2. Number of METIS runs in step 1 has order of $O(\log|V|)$. Worst-case number of iterations in step 2 is $|V|$, so the total number of METIS runs in A_{cl} can be estimated as $O(|V|)$. Every run of METIS has complexity of $O(|V| + |E| + k * \log(k))$ where k is the number of parts. Given $k \le |V|$, we get $O(|E| + |V| * \log(|V|))$ for the complexity of a single METIS run in A_{cl}. Therefore, the total complexity of METIS runs in A_{cl} has order of $O(|V| * |E| + |V|^2 * \log(|V|))$.

V. Experimental Study

The experimental study of the algorithms was performed on three sets of graphs. The goal of the experiments was to compare the greedy and clustering-based algorithms by value of objective function on different sets of input data.

Layered task graphs (Fig. 1) are typical for parallel data processing applications in such areas as digital signal processing (e.g. Cholesky decomposition, QR factorization [18], LU factorization [19], FFT [20]), multimedia, and parallel computations known as scientific workflows [21]. In such graphs, tasks on the previous layer transfer data to tasks on the next layer, with data transfers (edges) bypassing the layers only as exceptions.

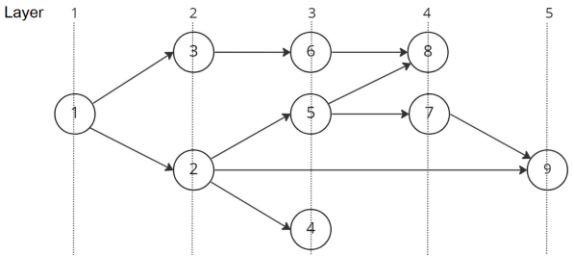

Figure 1. Layered graph.

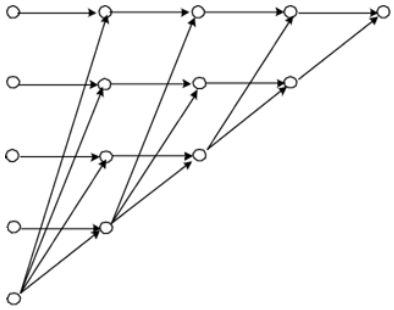

Figure 2. Triangle graph.

TABLE I. TASK GRAPH PROPERTIES IN THE EXPERIMENTAL DATA SET

Graph type	Number of nodes	Number of edges	Average density ($\|E\|/\|V\|$)	Basic task duration
Random	30 – 100	40 – 150	1.5	U{100, 400, 1600}
Layered	50 – 200	140 – 610	3	U(10, 50)
Triangle	55 – 465	90 – 870	1.8	U(10,100)

Triangle task graphs (Fig. 2) are a subset of layered task graphs. These graphs with regular structure represent the parallel implementation of Gauss-Jordan algorithm [22].

Random task graphs [23].

Layered and random task graphs were generated according to the schemes from [23]. The scheme for generation of triangle task graphs was quite straightforward due to their structure. Properties of the task graphs in the experimental data set are shown in Table I.

The input data set contained 100 random graphs, 361 layered graphs, and 50 triangle graphs.

Task graphs were allocated to following processor sets:

- processor groups with speed factors 3 and 2: 1 to 5 groups (2 to 10 processors in a set);
- processor groups with speed factors 4 and 1: 1 to 5 groups (2 to 10 processors in a set);
- processor groups with speed factors 5, 4, 3 and 2: 1 to 5 groups (4 to 20 processors in a set).

For every task graph and processor set, upper bound CR_U on the share of interprocessor edges (cut-ratio) was varied from 0.1 to 0.7 with step 0.05, and from 0.7 to 1.0 with step 0.1. This increase in step was made because with large CR_U the results of greedy and clustering-based algorithms were close to each other, with typical difference of objective function within 10% in favor of the latter algorithm.

A single experiment corresponds to a specific combination of task graph, set of processors, and value of CR_U. For such combination, the greedy algorithm A_{gr} and the clustering-based algorithm A_{cl} were run, with resulting objective function values of f_{gr} and f_{cl}. The result of the greedy algorithm was taken as a baseline, and the "advantage" of A_{cl} over A_{gr} (in %) was calculated:

$$Adv = 100\% * f_{cl} / f_{gr}$$

For example, Adv=60% means that A_{cl} produced a partition which is by 40% better than the partition produced on the same input data by A_{gr}.

With METIS using randomized partitioning algorithms, experiments reproducibility was guaranteed by supplying METIS with a fixed random seed. Preliminary experiments without fixed seed indicated the variation of objective function value in results of A_{cl} on the same input data within 5%, with rare outliers up to 10%. Similar scale of variation was indicated for results of A_{gr}. Variation of objective function is not to be messed with variation of Adv: e.g. for 10% variation of f_{cl}, invariable f_{gr}, and best Adv of 0.4, the worst Adv is 0.44, not 0.54.

Fig. 3 shows a typical set of heatmaps for Adv with processor sets listed above, and a random graph ("dag13") with 83 nodes and 123 edges. Horizontal axis shows the value of CR_U, vertical axis shows the processor set: e.g., 4_1x5 means 5 processors with speed factor 4, and 5 processors with speed factor 1.

In Fig. 3, we see the trends common to almost all the experiments. With loose CR constraint ($CR_U > 0.5$ for dag13) the advantage Adv is not significant. This can be attributed to good possibilities for A_{gr} to move the tasks between processors without getting into a dead-end. With strict CR constraint, Adv is not significant either, because A_{cl} cannot

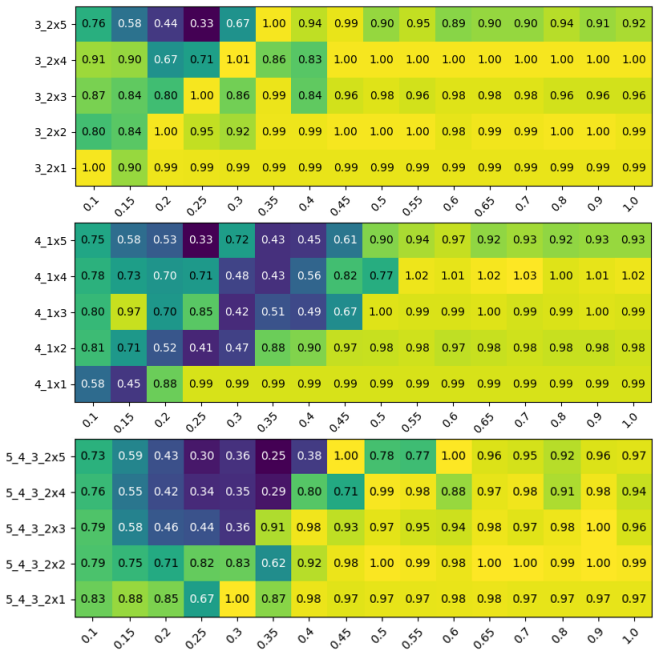

Figure 3. Advantage of A_{cl} over A_{gr} on a random graph (less is better).

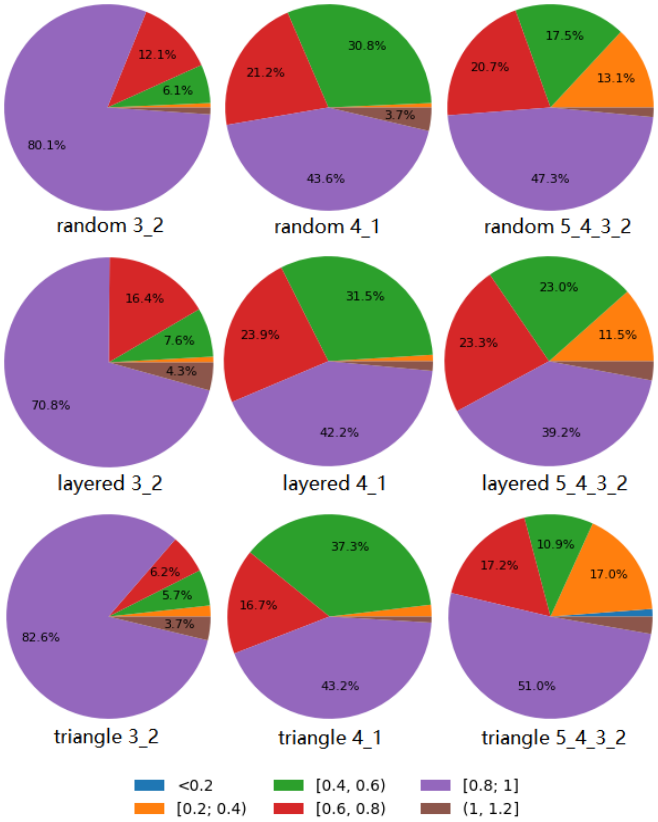

Figure 4. Advantage of A_{cl} over A_{gr}, per task graph class and set of speed factors. Less is better.

produce a good set of clusters: there are too few of them, or there are many, but some of them are too large and others too small. With "moderate" CR constraint, *Adv* becomes significant, with objective function for A_{cl} up to 4 times better than for A_{gr}. The advantage of A_{cl} over A_{gr} also becomes more significant with increase of the number of processors within the same set of speed factors. For processors with two speed factors, the more is the difference in their values (4 and 1 vs. 3 and 2), the greater is the need to change the initial allocation to offload the slow processors, and the greater is the effect of A_{cl} flexibility in such offloading, leading to more significant advantage over A_{gr}.

Fig. 4 shows the distribution of *Adv* to ranges for different graph types and processor sets. One pie chart corresponds to experiments with same graph type and set of speed factors. From the aggregated information in this figure, we can deduce that:

- There is a strong dependence of *Adv* on the set of processor speed factors (3 and 2 vs. 4 and 1) – the more is the difference between them, the more significant is *Adv*;
- With greater number of processors (twice more for the rightmost circles than for the others), the share of experiments with very significant (less than 0.4) *Adv* becomes noticeable: 13.1%, 11.5%, 17%;
- Dependence of *Adv* on the graph type (see circles in the same column) is much less than on the set of speed factors and on the number of processors.

There is a small share of experiments in which A_{cl} performs worse than A_{gr}: *Adv* is greater than 1, typically within 1.1, with several (less than 0.1%) instances between 1.1 and 1.2. This share corresponds to the cases when the initial partition for A_{gr} is better suited for per-task refinement than the one obtained by the cluster-based phase of A_{cl}. These rare layoffs can be compensated by running both A_{cl} and A_{gr} and choosing the best solution, which is not much longer than running only A_{cl}.

Implementation of the algorithms in Python and the input graph sets are freely available for academic use [24]. The implementation uses METIS and NetworkX free libraries.

The experiments were performed on the computer with following hardware: CPU Intel Core i7-10750H 2.6 GHz, 16 Gb RAM. Run times of A_{cl} were: up to 50 s with random graphs, up to 200 s with layered graphs, and up to 546 s with triangle graphs (see Table I for graph scale). A_{gr} finished in several seconds for every graph. Run time of A_{cl} can be reduced by parallel execution of the loop in step 2 of the algorithm with different values of j, as the iterations of this loop are independent.

Table II presents the information on A_{cl} run time depending on task graph scale. The information is provided for triangle graphs, which have more regular structure than random and layered graphs. Minimum and maximum times are taken for all experiments with graphs of the same size.

Dividing maximum A_{cl} run time by $|V|^2$, we get $5.95*10^{-3}$, $4.63*10^{-3}$, $2.38*10^{-3}$, $2.59*10^{-3}$ and $2.53*10^{-3}$ for 10, 15, 20, 25 and 30 layers, which gives an empirical estimation of A_{cl} complexity as $O(|V|^2)$ for triangle graphs and unchanging processor sets, where $|V|$ is the number of nodes in the graph. This is less than $O(|V|^2*\log(|V|))$ dependence on the number of nodes derived in the end of Section IV, possibly because the upper estimates for the number of METIS runs are not reached.

TABLE II. A_{CL} RUN TIME FOR TRIANGLE GRAPHS

Number of layers	Number of nodes	Number of edges	Min. run time, s	Max. run time, s
10	55	90	1.1	18.0
15	120	210	3.4	66.8
20	210	380	8.0	105.0
25	325	600	17.9	274.0
30	465	870	23.9	545.5

VI. CONCLUSION

Main contribution of this paper is the clustering-based algorithm for allocation of workload to heterogeneous processors under constraint on the share of inter-processor data transfers. The distinguishing feature of this algorithm is its ability to "work around" the obstacles to partition refinement, which are created by this constraint.

Future work includes:

- integration of this algorithm with scheduling algorithms described in [1];

- research of critical path-aware algorithms for workload allocation to heterogeneous processors;
- taking in account the network heterogeneity.

REFERENCES

[1] V. V. Balashov, V. A. Kostenko, I. A. Fedorenko, I. P. Savitsky, C. Sun, J. Gao, L. Zhou, and J. Sun, "Hybrid Algorithm for Multiprocessor Scheduling with Makespan Minimization and Constraint on Interprocessor Data Exchange," in *Proc. 2023 9th Internat. Conf. on Control, Decision and Information Technologies (CoDIT)*, IEEE, 2023, pp. 2525–2531.

[2] G. Karypis, "METIS and ParMETIS," in *Encyclopedia of Parallel Computing*, Springer, 2011, pp. 1117–1124.

[3] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., 1979.

[4] Ü. V. Çatalyürek, et al., "More Recent Advances in (Hyper)Graph Partitioning," *arXiv preprint arXiv: 2205.13202*, 2022.

[5] S. Wu and J. Hou, "Graph partitioning: an updated survey," *AKCE Internat. J. Graphs and Combinatorics*, vol. 20, no. 1, pp. 9–19, 2023.

[6] T. A. Ayall et al., "Graph Computing Systems and Partitioning Techniques: A Survey," *IEEE Access*, vol. 10, pp. 118523–118550, 2022.

[7] S. Schwartz, "An overview of graph covering and partitioning," *Discrete Mathematics*, vol. 345, p. 112884, 2022.

[8] S. Schwartz, "Architecture- and workload-aware graph (re)partitioning," Ph.D. dissertation, Dept. Comput. Sci., Pittsburgh Univ., 2017.

[9] N. Xu, B. Cui, L.-T. Chen, Z. Huang, and Y. Shao, "Heterogeneous Environment Aware Streaming Graph Partitioning," *IEEE Trans. Knowledge and Data Engineering*, vol. 27, no. 6, pp. 1560–1572, 2015.

[10] I. Moulitsas and G. Karypis, "Architecture aware partitioning algorithms," in *Proc. 8th Internat. Conf. on Algorithms and Architectures for Parallel Processing*, 2008, pp. 42–53.

[11] J. Xue, Z. Yang, S. Hou and Y. Dai, "When computing meets heterogeneous cluster: Workload assignment in graph computation," in *Proc. 2015 IEEE Internat. Conf. on Big Data*, 2015, pp.154–163.

[12] D. Kumar, A. Raj and J. Dharanipragada, "GraphSteal: Dynamic Re-Partitioning for Efficient Graph Processing in Heterogeneous Clusters," in *Proc. 2017 IEEE 10th Internat. Conf. on Cloud Computing*, 2017, pp. 439–446.

[13] M. Li, H. Cui, C. Zhou, and S. Xu, "GAP: Genetic algorithm based large-scale graph partition in heterogeneous cluster," *IEEE Access*, vol. 8, pp. 144197–144204, 2020.

[14] A. Jain, S. Sanyal, S.K. Das, and R. Biswas, "FastMap: a Distributed Scheme for Mapping Large Scale Applications onto Computational Grids," in *Proc. 2nd Internat. Workshop on Challenges of Large Applications in Distributed Environments*, 2004, pp. 118–127.

[15] V.T.N. Nguyen and R. Kirner, "Throughput-Driven Partitioning of Stream Programs on Heterogeneous Distributed Systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 27, no. 3, pp. 913–926, 2016.

[16] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, 1970.

[17] M.L. Islam, N. Nurain, S. Shatabda, and M.S. Rahman, "FGPGA: an Efficient Genetic Approach for Producing Feasible Graph Partitions," in *Proc. 2015 Internat. Conf. on Networking Systems and Security*, 2015, pp. 1–8.

[18] H. Casanova, J. Herrmann, and Y. Robert, "Computing the expected makespan of task graphs in the presence of silent errors," *Parallel Computing,* vol. 75, pp. 41–60, 2018.

[19] J. Kurzak, J. Dongarra, "Fully Dynamic Scheduler for Numerical Computing on Multicore Processors," LAPACK Working Note 220, pp. 1–10, 2009.

[20] M. Abdeyazdan, S. Parsa, and A.M. Rahmani, "Task graph pre-scheduling, using Nash equilibrium in game theory," *J. Supercomputing,* vol. 64, pp. 177–203, 2013.

[21] V. Prakash, S. Bawa, and L. Garg, "Multi-Dependency and Time Based Resource Scheduling Algorithm for Scientific Applications in Cloud Computing," *Electronics*, vol. 10, p. 1320, 2021.

[22] A. Gerasoulis and T. Yang, "On the granularity and clustering of directed acyclic task graphs," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 6, pp. 686–701, 1993.

[23] L.-C. Canon, M. El Sayah, and P.-C. Heam, "A Comparison of Random Task Graph Generation Methods for Scheduling Problems," in *Proc. 25th Internat. Conf. on Parallel and Distributed Computing*, 2019, pp. 61–73.

[24] Algorithm for heterogeneous partitioning. https://github.com/slonoul/heteropart/. Accessed 15 May 2025.