Flow-Partitioning within Hierarchical Process Mapping

ERBEN JAKOB, Heidelberg University, Germany

As the amount of communication in large computations increases it is increasingly crucial to find good algorithms which output a distribution of the suppresses such that communication costs are minimized. Especially if the cost of communications are irregular [11]. Therefore, it is important to know the architecture of the used system in order to map the different computational blocks on to processors in such a way, that the total cost of communication is held as low as possible. In our work we propose an alternation to the multilevel approach presented by Faraj et al. [10], which enables the flow oriented refinement method to be aware of the architecture of the system and the associated communication costs. However, while an easy theoretical advantage of this method could be shown this could not be reproduced in different experiments.

1 INTRODUCTION

The problem of efficiently performing a large computation with certain amounts of communication between different components can be viewed as two separate problems. First one needs to form blocks of tasks, where each block is executed on the same processor, this can be achieved by partitioning the computation graph. Secondly these blocks then need to be mapped to physical processors in the system. This mapping needs to be considerate of the architecture of the system, as any two processors may either bee in identical areas (i.e. they have a small distance) or they are far away from one another (i.e. they have a large distance). As recent research [2] however showed that an overall improvement in performance can be reached when both tasks not performed separately but aware of one another. The approach requires two make two assumptions about modern supercomputer systems which are generally valid. Firstly that communication patterns are sparse and secondly that there is a hierarchical representation of those patterns. In this hierarchical representation two nodes of the same level always have the same speed of communication. On this basis the multilevel approach was developed, which improved the process mapping task by integrating it into highly performant multilevel partitioning algorithms. The multilevel approach contracts the problem to a smaller problem and then can use partitioning algorithms to solve these compacted versions. One of the possible partitioning algorithms it uses is a flow partitioner which relies on a result from graph theory, that shows that the size of a minimum cut equals the size of a maximum flow. Hence, one can find a minimum cut, by determining the maximum flow.

2 PRELIMINARIES

2.1 Basic Concepts

Let $G = (V = \{0, \ldots, n-1\}, E)$ be an *undirected graph* with edge weights $\omega : E \to \mathbb{R}_{>0}$, vertex weights $c : V \to \mathbb{R}_{\geq 0}$, $n = |V|$, and $m = |E|$.

The *graph partitioning problem* is concerned with partitioning all $vinV$ into k distinct subsets called blocks, a solution which uses k blocks is called a k-partition. The quality of a solution is determined by the size of the created edge cut as well as the balance of the blocks. Let us denote the blocks with V_1,...,V_k. Note that by distinctiveness and completeness of the subsets, the following must hold: $V_1 \cup \cdots \cup V_k = V$ and $V_i \cap V_j = \emptyset$ for $i \neq j$. The *balancing constraint* ensures that the weight of a block (the sum of weights

Author's address: Erben Jakob, jakob.erben@stud.uni-heidelberg.de,Msc.Data&ComputerSience,3727893, Heidelberg University, Im Neuenheimer Feld 205, Heidelberg, Baden-Württemberg, Germany, 69120.

of a blocks vertices) does not exceed a threshold respecting a balancing factor ϵ. Hence, for $i \neq j : c(V_i) \leq c(V_j) * (1 + \epsilon)$. The *edge-cut* of a k-partition measures the weight of all edges incident to two vertices of different blocks. It can be expressed as $\sum_{i<j} \omega(E_{ij})$, where $E_{ij} := \{\{u, v\} \in E : u \in V_i, v \in V_j\}$.

For a k-partition we call a pair of blocks *neighboring blocks* if they are connected by some edge. Any node $v \in V_i$ which has a neighbor $w \in V_j, i \neq j$ is part of the boundary its block and is hence called *boundary node*.

In order to perform a flow partitioning we need to change our representation of a graph slightly, by defining a corresponding network for which the problem can be solved. Therefore, let $G_f = (V_f, E_f)$ be a flow graph. Starting from a graph G we added two additional vertices called *source* and *train* (short: s, t) to the vertices (i. e., $V' = V \cup \{s, t\}$). All edges are directed meaning $e = (u, v)$ is an edge from vertex u to vertex v and each edge has a non-negative capacity assigned with it ($i.e., cap(u, v)$). A flow function is a function on the edges of a network (which can now be interpreted as a tuple of vertices). A function $\{f : V_f \times V_f \rightarrow \mathbb{R}_{>0}\}$ is a flow if it fulfills three constraints. It has to satisfie a *capacity constraint* ($i.e., f(u, v) \leq$), a *symmetric constraint* ($i.e., f(u, v) = -f(v, u)$) and *flow conservation constraint* in all vertices except s and t ($i.e., \left\{\sum_{v \in N(u)} f(u, v) = 0 : \forall v \in V' \setminus \{s, t\}\right\}$). Ford and Fulkerson showed in 1956 that the value of a flow $|f| = sum_{v \in V_f f(s,v)}$ if it is maximal ($i.e., |f| \geq |f'|$ for any st-flow f') is identical to the weight of a minimal *ST-cut*. A minimal cut is a set of edges S which separates s from t, such a set is minimal if the sum of its edges capacities is minimal.

Assume that we are partitioning n processes for a topology containing k processing elements (PEs). Let $\mathcal{C} \in \mathbb{R}^{n \times n}$ denote a matrix indicating the communication volumes and let $\mathcal{D} \in \mathbb{R}^{k \times k}$ denote the (implicit) topology matrix or distance matrix. For two processes i and j the amount of communication between them is denoted with $\mathcal{C}_{i,j}$. Say they are assigned to PEs x and y then the distance matrix gives a cost per communication between the two PEs $\mathcal{D}_{x,y}$. Therefore, the total cost of communication can be expressed as $\mathcal{C}_{i,j}\mathcal{D}_{x,y}$.

Based on the assumption, that supercomputers are hierarchically structured, the structure of the system is described with a hierarchy sequence $\mathcal{S} = a_1 : a_2 : ... : a_\ell$. It indicates that each processor has a_1 cores, each node a_2 processors, each rack a_3 nodes, and so forth. $k = \Pi_{i=1}^{\ell} a_i$ hence expresses the total number of PE's in the system. Similarly, $D = d_1 : d_2 : \ldots : d_\ell$ is a sequence which describes the communication cost between the different hierarchy levels. Two cores in the same processor communicate with cost d_1, two cores in the same node but in different processors communicate with cost d_2, two cores in the same rack but in different nodes communicate with cost d_3, and so forth.

2.2 Related Work

There has been a lot of work on problem of graph processing, for further information we refer to [3] [4] [12]. There have been different approaches to the task of process mapping in recent years. Kirchbach et al. [9] tackled the problem by splitting the problem into a graph partitioning a subsequent mapping of said partitions onto the different PEs. However, it was shown that by Berti and Träff [2], that such a division of the tasks could lead to worse results.

M. F. Faraj, A. van der Grinten, H. Meyerhenke et al. [10] extended the multilevel graph partitioner to also be able to solve the process mapping problem. They therefore could consider both the partitioning and the mapping at the same time, as well as make use of the good results of the multilevel partitioner.

Predari, Tzovas, Schulz et al. [11] used an MPI-based algorithm, to solve the process mapping issue in a parallelized fashion. It combines the parallel partitioner ParHIP and parallel label propagation mapping in order to receive, to get a fast but still relatively high quality process mapping solution.

The flow partitioning uses the equality of the size of a minimum cut and s maximum flow for proof of this equality we refer the reader to [7]. For an efficient implementation of the push relabel algorithm we want to point to [5] as well as the paper introducing the algorithm [8].

3 DISTANCE-AWARE FLOW PARTITIONING

We will proceed this explanation in three steps. Firstly the general flow partitioning approach is described, secondly a theoretical example is shown for which the communication unaware version performs poorly. Lastly the extension, which enables the consideration of the explicit communication costs, is explained. The flow partitioner is used during refinement the multilevel algorithm has already computed a partition and after the uncontraction the quality of the solution is refined.

Flow partitioner

Call the two already existing partitions L and R we can construct the network for computing the flows as follows: $V_f = L \cup R \cup s, t$ and $E_f = E_{L,R} \cup \{sl : s, l \in L\} \cup \{rt : r \in R, t\}$ where $E_{L,R}$ is the set of edged between the nodes in L and R. Please note that all edges connecting some vertex to either s or t are assigned infinite capacity, such that they are never fully saturated and hence can never be part of a minimal st-cut. A St-cut in this network would give a partition of the nodes of the nodes into L' (all vertices reachable from s) and R' such that the edge cut minimal if it is identical to the maximum flow. The overall process would now work as follows: Take two given partitions and create a network. Secondly compute a maximum flow. This could be done with a few different algorithms, the given implementation uses the *Push-Relabel algorithm* to find the maximum flow. After the maximal flow was determined, a set of edges with same sum of weights can be found and the one which minimizes the cut while giving the best balance is returned. While this approach has been used successfully used in graph partitioning it can run into issues for process mapping as the next example will show.

Example problem distance unaware partitioner

This version of a flow partitioner has however one shortcoming, as it is not aware of the distances between PEs. Lets for example assume there are 4 partitions and corresponding PEs (A, B, C and D). Let's assume all edges between individual tasks in the graph have the same amount of communication and hence a weight of one. Say $e(A, B)$ is the number of edges between partition A and B and as all edges have weight one this is identical to the amount of communication (not considering distance) between any two $\sum_{a \in A, b \in B} \mathcal{C}_{i,j} = \sum_{\{a \in A, b \in B\,:\,(a,b)=e\}} \omega(e)$.

Now assume that $e(A, B) = e(A, C) = e(B, C) = 3$ and that $\mathcal{D}_{A,B} = \mathcal{D}_{A,C} = 1$ and that $\mathcal{D}_{B,C} = 10$. This configuration gives a cost total communication cost of $e(B, C) * \mathcal{D}_{B,C} + e(A, B) * \mathcal{D}_{A,B} + e(A, C) * \mathcal{D}_{A,C} = 3 * 10 + 3 * 1 + 3 * 1 = 36$. Assume now that the flow partitioner moves two vertices from A to B giving A' and B' and with $e(A', B') = 1$. Then it initially seems like the partitioner decreased the total amount of communication by two. However, further assume that one of the moved vertices was also connected to C. Hence get

$e(A', B') = 1, e(A', C) = 2$ and $e(B', C) = 4$. With the same distances as before this gives a total cost of $e(B', C) * \mathcal{D}_{B,C} + e(A', B') * \mathcal{D}_{A,B} + e(A', C) * \mathcal{D}_{A,C} = 4 * 10 + 1 * 1 + 2 * 1 = 43$ and the total cost has actually increased to by seven.

Distance aware partitioner

To avoid these cases this work introduces a version of the flow partitioner which is aware of the actual cost of moving vertices in between partitions. This is archived by two modifications to the network model. Firstly the capacity function ω is modified to also consider the distance between the two partitions under consideration. Secondly additional edges are introduced to consider the true change in communication cost when moving a vertex in between partitions.

To make the actual communication cost more pronounced we multiply all capacities of edges not incident to s or t, with the distance between the two PEs associated with partitions under consideration. Assume again that all weights are 1 and that $\mathcal{D}_{A,B} = 3$. Then a minimal cut with three edges would have an accurate weight of nine. Please note that this change alone will not change the outputs of the flow problem as it scales all edge weights uniformly and hence simply scales the problem itself.

To correctly consider the change in communication cost when partitioning we introduce two additional edges for each vertex ($\neq s, t$). One of which goes from s to any vertex and one which goes from that vertex to t. These edges encode 2 different information depending on the original position of the vertex. Let's look at any $l \in L$ then the edge from it to t encodes the current cost of communication to vertices not in part of the considered partitions. Hence, if the algorithm decided to cut this edge then l would be part of L_{new} then the cost of cutting this edge would represent the communication associated with the outside communication with l. Similarly, for some vertex $r \in R$ the same information about the current communication cost is represented in an edge to s. All vertices have a second edge encoding the potential cost with moving the vertex to the other side for all l inL those edges start in s, and for all $r \in R$ they end in t. Say the algorithm decides to cut one of those edges for a left vertex then this resembles the outside information associated with a partitioning where l is part of R'. There are two sets of newly created edges are called *communication correction edges drain* (CCD) and *communication correction edges source* (CCS). The weight of these edges can be described as follows:

$$\text{CCS: } \forall l \in L : cap(s, l) = \sum_{x \in N(l) \setminus \{L \cup R\}} \mathcal{D}_{R,pe(x)} * \mathcal{C}_{l,x}$$

$$\text{CCD: } \forall l \in L : cap(l, t) = \sum_{x \in N(l) \setminus \{L \cup R\}} \mathcal{D}_{L,pe(x)} * \mathcal{C}_{l,x}$$

Similarily, the weights can be defined for edges adjacent to the right partition.

Going back to the example from the last subsection it is impossible for the distance aware flow partitioner to move vertices in a way such that the total weight of communication increases instead of decreasing, as it processes all needed information to avoid these mistakes.

4 EXPERIMENTS

We implemented our work within the MultiLevelMap Framework (C++) which is based on the KaHIP framework. We compiled the project with compiler version [MISSING] and the following optimization settings [Missing]. The used machine was running Ubuntu 22.04 and the kernel version 5.19.0-32. It contained an 8-core / 16-thread Ryzen 7 5825U processor clocked at up to 4.5 GHz and was equipped with a total of 16 gigabytes of RAM. All tests

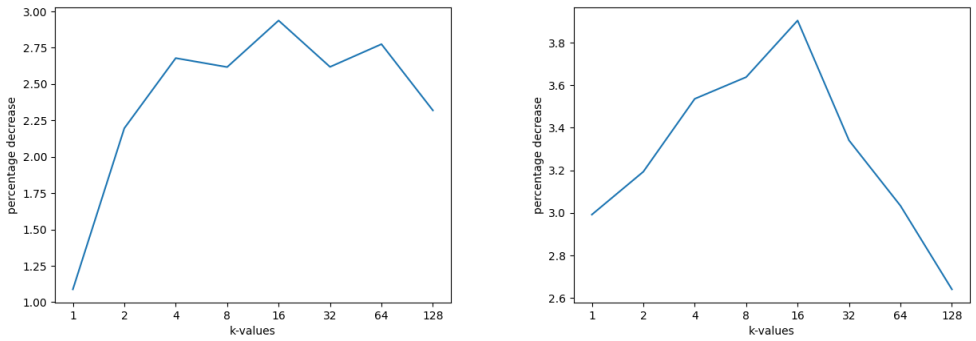
allowed for a total of 3% of imbalance and were tested for different partition counts. These depended on the hierarchy describing sequence the following sequences were used. Let the set of test-sequences be $S' = \{\{4 : 16 : 64 * k\} : k \in \{1, 2, 4, 8, 16, 32, 64, 128\}\}$. This resulted in three different hierarchical levels which were assigned the following distance sequence $D = \{1 : 10 : 100\}$. Lastly a fixed seed was used in order to keep results as reproducible as possible.

4.1 Dataset

The graphs for testing the algorithm come from two sources. Firstly the five largest graphs from Chris Walshaw's benchmark archive [13] were used. All further graphs were originally part of the 10th DIMACS Implementation Challenge [1]. All rgg_X are random geometric graphs with 2^X nodes. Each node can be interpreted as a random point in the unit square two nodes are connected if the Euclidean distance between them is below 0.55 $\frac{ln(n)}{n}$. The graphs del_X are so-called Delaunay triangulation of 2^x random points in the unit square. There are some graphs form the numerical graph section of the DIMACS benchmark set (i.e. af-shell9, af-shell10 ...). The graphs europe, asia, great-britain, belgium and germany are large road networks of the respective areas [6]. Table 1 shows the basic properties of the graphs that were used.

Graph-Instance	n	m	Graph-Instance	n	m
Walshaw instances					
auto	∼450k	∼3.3M	144	∼145k	∼1M
bcsstk30	∼28k	∼1M	bcsstk32	∼45k	∼100k
m14b	∼215k	∼1.6M			
road-network instances					
asia	∼12M	∼13M	belgium	∼1.4M	∼1.5M
europe	∼2M	∼6.5M	germany	∼11.5M	∼12M
great-britain	∼7.7M	∼8.1M			
numerical instances					
af-shell9	∼500k	∼8.5M	af-shell10	∼1.5M	∼25M
audikw1	∼940k	∼38M	ecology-1	∼1M	∼2M
ecology-2	∼1M	∼2M	G3-circuit	∼1.5M	∼3M
kkt-power	∼2M	∼6.5M	ldoor	∼950k	∼23M
nlpkkt120	∼3.5M	∼46M			
delauny and random geometric instances					
delauny10	∼1M	∼1.9M	delauny15	∼32k	∼98k
delauny24	∼16M	∼50M	rgg15	∼32k	∼160k
rgg23	∼8.3M	∼63M			
various instances					
vsp-barth5	∼32k	∼100k	vsp-beref	∼14k	∼98k
citation	∼270k	∼1.1M	coAuthers	∼300k	∼980k

Table 1. Tested graph-instances



(a) Percentage increase in final objective- - strong configuration

(b) Percentage increase in final objective- - social configuration

Fig. 1. Comparison distance aware flow partitioner vs. baseline implementation of MultiLevelMap

4.2 Comparison against state of the art

In a first experiment the results for the final objective between the standard algorithm used by MultiLevelMap and a version which uses a distance aware flow partitioner are compared. Image 1a shows the percentile increase for the different k-values averaged over the different graphs for the strong configuration of the MultiLevelMap algorithm. Image 1b shows the same result for the social configuration. In both cases the distance aware flow partitioner performs worse than the standard Fibonacci- Mathesis multiway partitioner. The results are about 3% worse than the baseline.

4.3 Comparison against flow partitioner

Additionally, the distance aware flow partitioner was compared against the standard version. Image 2 shows the percentage decrease in the final objective value compared over all instances. One can see that it alternates around 0% which tells us two information. Firstly there are instances where the distance aware partitioner leads to a better solution and hence its

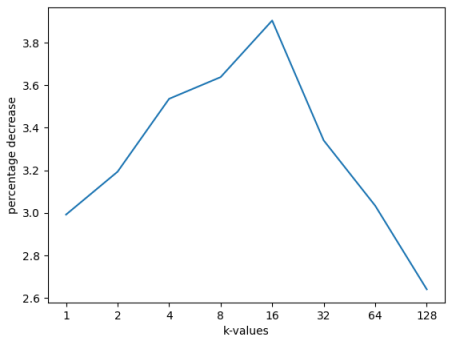


Fig. 2. Percentage increase in final objective- - social configuration

theoretical benefits can in some cases be used. Secondly it appears that the MultiLevelMap algorithm produces good results regardless of distance awareness as the partition quality seems to be deciding factor in the final objective score.

5 CONCLUSION

While the distance aware flow partitioner seems like a good idea from a theoretical standpoint it did not lead to a success in implementation as the experiments determined that it did not significantly lead to better results than the standard flow partitioner, which meant that it could not replace the standard partitioning solution of the MultiLevelMap algorithm. Despite all of this it could be possible that the distance aware flow partitioner could lead to improved results in some other configuration or some other problem, where a secondary factor determines the true result of a cut.

REFERENCES

[1] D. A. Bader, H. Meyerhenke, P. Sanders, C. Schulz, A. Kappes, and D. Wagner. Benchmarking for graph clustering and partitioning. In *Encyclopedia of Social Network Analysis and Mining*, pages 73–82. Springer, 2014.

[2] Guntram Berti and Jesper Larsson Träff. What mpi could (and cannot) do for mesh-partitioning on non-homogeneous networks. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 13th European PVM/MPI User's Group Meeting Bonn, Germany, September 17-20, 2006 Proceedings 13*, pages 293–302. Springer, 2006.

[3] Charles-Edmond Bichot and Patrick Siarry. *Graph partitioning*. John Wiley & Sons, 2013.

[4] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. *Recent advances in graph partitioning*. Springer, 2016.

[5] Boris V Cherkassky and Andrew V Goldberg. On implementing the push—relabel method for the maximum flow problem. *Algorithmica*, 19:390–410, 1997.

[6] D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering route planning algorithms. In *Algorithmics of Large and Complex Networks*, volume 5515 of *LNCS State-of-the-Art Survey*, pages 117–139. Springer, 2009.

[7] Reinhard Diestel and Reinhard Diestel. Flows in networks. *Graph theory*, pages 149–154, 2017.

[8] Andrew V Goldberg and Robert E Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.

[9] Konrad Von Kirchbach, Christian Schulz, and Jesper Larsson Träff. Better process mapping and sparse quadratic assignment. *Journal of Experimental Algorithmics (JEA)*, 25:1–19, 2020.

[10] H. Meyerhenke et al. M. Faraj, A. van der Grinten. High-quality hierarchical process mapping. *CoRR*, 2020.

[11] Christian Schulz Maria Predari, Charilaos Tzovas and Henning Meyerhenke. An mpi-based algorithm for mapping complex networks onto hierarchical architectures. pages 167–182, 2021.

[12] Christian Schulz and Darren Strash. Graph partitioning: Formulations and applications to big data. In *Encyclopedia of Big Data Technologies*, pages 1–7. Springer, Cham, 2018.

[13] A. J. Soper, C. Walshaw, and M. Cross. A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph-Partitioning. *Global Optimization*, 29(2):225–241, 2004.