2015 The 5th International Workshop on Computer Science and Engineering
doi: 10.18178/wcse.2015.04.020

Complexity Based Load Balancing for Distributed Branch and Bound Method

Bo Tian, Mikhail Posypkin

Lomonosov Moscow State University, Institute for Information Transmission Problems of RAS

Keywords: Branch-and-Bound, Parallel Computing, Load Balance.

Abstract. We proposed a new static load distribution strategy for parallel Branch-and-Bound method based on complexity estimates of sub-problems appearing in a Branch-and-Bound tree. This strategy can be used on parallel systems with low connectivity where dynamic load balancing is problematic. The experimental results demonstrated the superiority of the proposed strategy over other three strategies under test. Based on these results we draw some conclusions about the duration of the first (serial) part of the algorithm and choice of a load distribution strategy.

Introduction

The main source of computing power in modern HPC systems is a growing number of simultaneously working processors (cores). However supporting fast and reliable communication among thousands of parallel processors is a difficult task that can significantly increase the cost of the equipment. That is why many HPC systems e.g. volunteer grids, map-reduce clusters, general purposed graphical processing units (GPGPU) provide limited or no communication among parallel processors. The mentioned systems are from completely different areas but they share the common approach to communication: the processing is done by independent threads (processes) that rarely communicate to exchange data.

Therefore the absolute majority of parallel implementations of Branch-and-Bound (B&B) methods heavily rely on dynamic load balancing which is not a problem in a tightly connected HPC system. However systems with limited connectivity need a different approach based on static work distribution: the workload should be divided into several independent work units before distributing among parallel processors to avoid communications during the computational process.

If all processors have similar performance even partitioning provides best load balancing. However dividing branch-and-bound tree into equal sub-trees is problematic since the size and the structure of a tree is not known in advance. In this paper we propose a new load distribution strategy based on modified round-robin scheduling applied to a list of sub-problems ordered according to descending complexity. Several existing static load distribution strategies including the proposed one are evaluated on various test cases and compared. We show that when we can reliably estimate the complexity of sub-problems the new strategy outperforms others. If performance prediction is problematic random strategy seems to be the best option.

Background and Related Works

The methods considered in this paper are aimed at the following global optimization problem

$$f(x) \to \max \ s.t. \ x \in X. \quad (1)$$

One of the most popular approach to solve problem (1) is the Branch-and-Bound (B&B) method [1] which is a generic name for algorithms to split an initial problem into sub-problems which are sooner or later eliminated by bounding rules. In B&B theory splitting is traditionally called branching. Branching rules define how the sub-problems are separated into smaller ones. Bounding rules determine whether a sub-problem can yield a solution better than the best solution found so far. The latter is called an incumbent solution. If the bounding rule failed the sub-problem is discarded from the further search.

For many practical problems Branch-and-Bound methods require the amount of computing resources beyond the power of a single-CPU workstation. Fortunately this family of methods is highly suitable for parallel and distributed computing: after splitting sub-problems can be processed simultaneously.

The ultimate goal of B&B parallelization is the reduction of the *makespan*: the difference between the start and the end of the computations. The main issue preventing the reduction of the makespan is the load imbalance – a situation when processors have to wait each other for tasks to process due to the uneven load distribution.

In approaches for tightly coupled shared memory multiprocessors [2, 3] threads cooperate by adding sub-problems to or removing sub-problems from the shared pool. To reduce the number of synchronizations they also maintain their private pools. The parallel B&B algorithm stops when all pools are empty. These schemes use intensive communication among processors to ensure uniform work distribution among pools.

The implementation of B&B on a shared memory system is relatively simple, since each processor can access the shared memory and a number of cores is small. Implementation of B&B algorithms on the distributed environment was also studied extensively in the literature, see [4, 5] for surveys. These approaches use intensive message-passing to achieve good load balance among parallel processors.

Highly suitable for tightly coupled systems like shared-memory multi-processors and HPC clusters the mentioned approaches are not applicable to systems with low connectivity like grids or GPU cards. Recently some approaches for this sort of systems were presented [7]. The algorithm SelfSplit proposed in [6] forces all processors to perform same sequence of the initial steps. Then each processor takes unsolved sub-problems assigned to it for further processing and solves them to the end. After that obtained results are collected on the master processor and the best solution is returned. We further develop this approach and improve it with new load balancing strategy based on reversing traditional round robin procedure. Also we present a careful experimental evaluation of different load balancing strategies and draw some conclusions.

Outline of the Approach

1.1 General scheme

Consider a hypothetical distributed system consisting from one dedicated node called server and a number of client nodes. The nodes may be any sort of processors, e.g. GPGPU cores, CPU cores in an HPC cluster, or volunteer grid client PCs. The proposed algorithm has three subsequent phases (Figure 1). During the first phase server performs a number of B&B steps to generate a frontier: a set of sub-problems not yet split by the branching rule or eliminated by the bounding rule. Then sub-problems from the frontier are packed into workunits and workunits are sent to clients (one workunit per client). During the second phase clients process sub-problems from the received workunits to the end. The third phase starts when all clients finished their jobs. At the third stage the incumbents are collected by the server and the best result is reported.

The proposed scheme is extremely suitable for computational environments with low connectivity because communication is required only between first and second phases and to collect results. Moreover the initial communication can be skipped by repeating the first phase on clients. This strategy saves resources when the clients are allocated at the same time as the server. This is the case for GP GPUs and computational clusters running batch processing systems. Otherwise it is more efficient to send sub-problems. Examples of such systems are desktop grids where clients can join or leave computations at any moment.

The goal is to minimize the makespan. For the sake of simplicity we assume communication time be negligible. Then makespan can be computed as follows $Msp = T_S + T_C$, where T_S represents the time of the first phase (server part) and T_C is the time of the second phase (client part). Clearly T_C is equal to the largest running time of a workunit:

$$T_C = \max\ t(WU_j) \quad j = 1, ..., w.$$

where $t(WU_j)$ is the running time of j[th] workunit.

Though the proposed strategy looks very simple it raises some important questions:

i) Which branching strategies to use on the server and on clients?

ii) When to stop the first phase?

iii) How to pack sub-problems to workunits?

In the sequel we'll answer these questions.

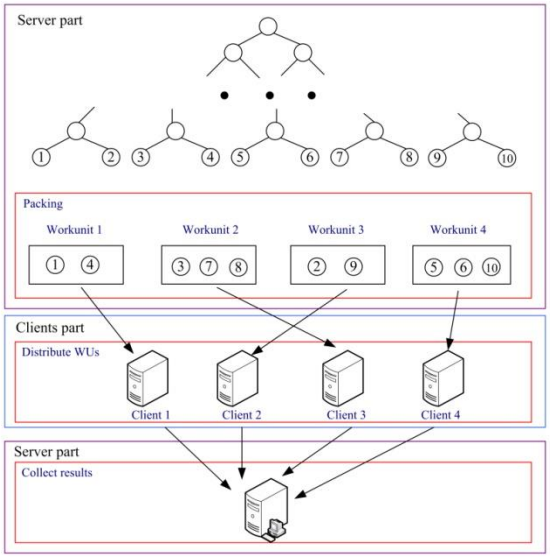

Fig.1: General scheme of the proposed approach

1.2 Branching strategy

We consider two search strategies of tree traversal: DFS (Depth First Search) and WFS (Width First Search). Starting from the tree root DFS follows one of the branches of the tree as far as possible and then returns back, i.e. it always chooses one of the nodes most distant from the tree root. WFS always chooses one of the nodes closest to the tree root. In our approach at the first phase the server uses the WFS strategy because it allows generating a large frontier in a relatively small number of steps. This is needed to provide an adequate amount of sub-problems for assigning to the clients. The server stops running the sequential WFS solver until the amount of sub-problems reaches an upper threshold, or the given number of iterations is done. The client node solves sub-problems in the workunit by a sequential DFS solver since DFS requires much less memory and therefore normally runs faster.

1.3 The duration of the first phase and packaging strategy

With w clients, a simple parallel scheme locally splits the initial problem into w sub-problems, send the sub-problems to different nodes, then wait for them to finish. Because a typical B&B tree is unbalanced, some machines will complete their sub-problems much faster than the others. Since T_C is determined by the largest running time of a most complex workunit the goal is to minimize this time, i.e. to make workunits as balanced as possible. Clearly the lower bound for T_C is attained when all workunits are equal.

To distribute the work uniformly each workunit receives $\lfloor s/w \rfloor$ or $\lfloor s/w \rfloor + 1$ sub-problems: the first $(s \bmod w)$ workunits have $\lfloor s/w \rfloor + 1$ sub-problems in each, the remaining workunits have $\lfloor s/w \rfloor$ in each.

In our paper, we implemented four different workunit packaging strategies. In a *dense strategy* workunits are produced by packing sub-problems neighboring in the frontier. The advantage of this

strategy is that it allows a straightforward and efficient implementation. Indeed the natural implementation of the WFS strategy is the FIFO queue. It's easy to see that adjacent sub-problems in the B&B tree reside close to each other in this queue. The dense strategy produces workunits by simply pulling sub-problems from one of the ends of this queue. The weak point of this strategy is that workunits stores adjacent tree nodes which usually have similar complexity: "hard" sub-problems are grouped together producing a significant load imbalance.

The random strategy copes with the adjacent nodes problem by using Fisher Yates shuffle to generate a random permutation of a set of sub-problems before applying the dense strategy. The idea of random strategy is that by shuffling the frontier we achieve more uniform distribution of the sub-problems complexity than without it.

The two remaining strategies are based on complexity estimates. The idea is that we provide a way to somehow estimate the complexity of the sub-problems and then use these estimates to produce workunits. Both strategies belong to the class of cyclic distributions (explained later). For the sake of simplicity assume that the number of jobs is evenly dividable by the number of workunits: $s = m \cdot w$. A cyclic distribution strategy starts from sorting sub-problems in an order of decreasing complexity estimates. The distribution is completely defined by a set of n permutations $\sigma_1,...,\sigma_n$ of a set $1,..,w$. The workunit i gets sub-problems with numbers $\sigma_1(i), w+\sigma_2(i),...,(m-1)w+\sigma_n(i)$. The cyclic distribution strategy can be conveniently visualized as a table where columns represent workunits (see figure 2).

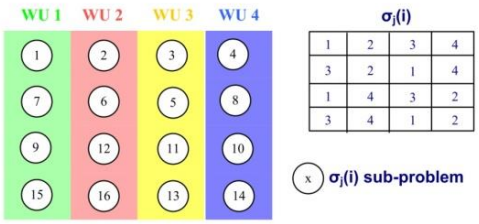

Fig.2: Cyclic distribution strategy (4 workunits, 16 sub-problems)

Below we consider two variants of the cyclic distribution: normal round robin (NRR) and reverse round robin strategies (RRR). In NRR sub-problems are equidistantly picked from the sorted list and packed in the final result queue. All permutations for this strategy are identical mappings:

$$\sigma_j(i) = i,\ i = 1,...,w,\ j = 1,...,n$$

The NRR strategy is very popular and is used as a default in many systems, e.g. [6]. However this strategy has the worst performance from all cyclic distributions. Indeed, the last (with number w) workunit contains the most complex sub-problems from each row, thus reaching the upper bound of the makespan. Reverse round-robin strategy (RRR) inverts each row. In this strategy the load distribution in rows is more even because most complex sub-problems in one row are combined with easiest sub-problems from another.

We identified two particular cases when RRR gives the lowest running time from all cyclic distributions: when there are only two rows (n=2) and when the complexity is a linear function of the amount of sub-problem and the number of rows is even. For a general case RRR is not necessary the best cyclic distribution. Experimental evaluation shows that RRR provides reasonable performance and remarkably outperforms NRR.

Experimental Evaluation

To test our approach we need a global optimization problem solvable by a Branch-and-Bound method where the complexity of sub-problems can be easily estimated. We selected the subset-sum problem which complexity is well studied [8]. The classical SSP is defined as follows: given a set of items with given weights and a knapsack select a subset of the items whose total weight is closest to the knapsack capacity, without exceeding it. Bounds obtained in [8] and our experiments suggest

that when the set of items is fixed the closer the capacity to the half of the sum of item weights the larger complexity. Since we generate the frontier where all sub-problems reside on the same layer and therefore have the same set of free items the complexity estimate is just the distance between the remaining capacity and the half of the total weight of the free items.

The goal of experiments was two-fold: to evaluate the efficiency of the proposed packaging strategies and to identify the optimal duration of the first Phase. The server works similar for all strategies: it performs WFS steps until the amount of sub-problems reaches $f \cdot w$ where f is a fixed factor. In essence factor is a minimal number of sub-problems in a workunit. After that server continues WFS until all sub-problems in the frontier have the same amount of free items (reside on the same tree layer).

In the first experiment we generated 10 random subset sum instances with 30 items. The number of workunits was equal to 1024 and we set $f = 2$.

Figure 3 compares makespan for four load balance strategies under consideration. Instead of measuring real time we collected the number of B&B steps (iterations) which under normal circumstances is proportional to the running time. However unlike running time the number of steps is not affected by various random events and can be precisely measured.

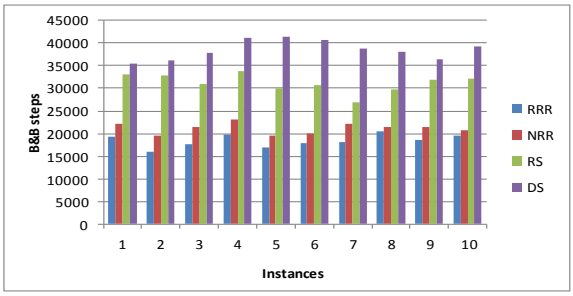

Fig.3: Makspan of 4 strategies in 10 random instances

From this experiment it clearly follows that the cyclic distributions significantly (in approximately 1.5-2.5 times) outperform random and dense strategies and Reverse Round Robin always give better results w.r.t. Normal Round Robin. However it's not clear how many sub-problems should be generated by the server. Intuition suggests that generating more sub-problems provides better opportunities for load balancing but can slow-down the computations due to the long serial Phase 1. Thus we have to stop as soon as we get a sufficient amount of sub-problems for efficient load-balancing.

Table 1 presents the value of the load balance metrics $L_b = t_l / t_a$ for various factors. Here t_l is a running time of the most complex workunit and t_a is an average running time of all workunit. The less the value of the metrics - the better load balance. Perfect load balance means $L_b = 1$. The obtained results show that the increasing factor from 4 to 8 gives a little improvement of the load balance which is already close to ideal. So it seems 4 is already a sufficient value of the factor and there is no sense the increase it further.

Surprisingly dense and random strategies show no improvement with increasing of the factor. It means that for these strategies server should stop as soon as it produces the number of sub-problems equal to the number of workunits.

Table 1: Load balancing on different amount of workunits and layer

Workunits	Factors	L_b			
		RRR	NRR	RS	DS
16	1	1.3148351	1.314835137	1.314835137	1.314835137
	2	1.169986	1.278234366	1.349550228	1.388454663
	4	1.0720355	1.166923884	1.382837537	1.482735334
	8	1.0099226	1.096536207	1.349837294	1.581215522
64	1	1.4836763	1.483676343	1.483676343	1.483676343
	2	1.1680544	1.363137438	1.583109795	1.585560332
	4	1.0336766	1.214525381	1.621355777	1.702615037
	8	1.0166067	1.083531563	1.700568635	1.84284239
256	1	1.7028908	1.702890812	1.702890812	1.702890812
	2	1.1082803	1.470869619	1.803547691	1.896096912
	4	1.0294256	1.187945375	1.949818237	2.033673821
	8	1.0261474	1.072008158	2.402864596	2.311203946
1024	1	1.8923109	1.89231094	1.89231094	1.89231094
	2	1.1280052	1.59749536	2.109386973	2.012984598
	4	1.032372	1.316665408	2.360545497	2.424957022
	7	1.0302877	1.167014988	2.769827465	2.981578639

Conclusion

We proposed a new static load distribution strategy for parallel Branch-and-Bound method based on complexity estimates of sub-problems appearing in a B&B tree. This strategy can be used on parallel systems with low connectivity where dynamic load balancing is problematic. The performed experiments demonstrated the superiority of the proposed (Reverse Round Robin) strategy w.r.t. other static load distribution strategies. Experimental results showed that generating 2-4 times more sub-problems than workunits improves load balance for cyclic distribution but gives no improvement for dense or random load-balancing strategies. In future we are going to consider more global optimization problems and performed experiments on real parallel systems.

Acknowledgements

The research work was supported by the program of China Scholarships Council (No.201308090004).

References

[1] Lawler E. L., Wood D. E. Branch-and-bound methods: A survey, *Operations research*, T. 14, №. 4, pp. 699-719, 1966.

[2] B. Mans, T. Mautor and C. Roucairol, A Parallel Depth First Search Branch and Bound for the Quadratic Assignment Problem, *European Journal of Operational Research (Elsevier)*, No. 81(3), pp. 617-628, 1995.

[3] Evtushenko, Yu., Posypkin, M., Sigal, I.: A framework for parallel large-scale global optimization. *Comput. Sci. Res. Development* 23(3), pp. 211-215, 2009.

[4] Ananth G, Kumar V, Pardalos P, Parallel processing of discrete optimization problems. Encycl of Microcomput 13, pp.129-147, 1993.

[5] Crainic T, Cun B, Roucairol C, Parallel branch and bound algorithms. In: Parallel combinatorial optimization, Chapter 1, JohnWiley & Sons, New Jersey, pp. 1–28, 2006.

[6] M. Fischetti, M. Monaci, D. Salvagnin, "Self-splitting of workload in parallel computation", Integration of AI and OR Techniques in Constraint Programming (CPAIOR), *Lecture Notes in Computer Science*, Volume 8451, pp. 394-404, 2014.

[7] Melab N. et al. A GPU-accelerated branch-and-bound algorithm for the flow-shop scheduling problem //Cluster Computing (CLUSTER), 2012 IEEE International Conference on. – IEEE, pp. 10-17, 2012.

[8] R. M. Kolpakov, M. A. Posypkin, Upper and lower bounds for the complexity of the branch and bound method for the knapsack problem, Discrete Mathematics and Applications, No. 20(1), pp. 95-112, 2010.