Procedia Computer Science

Volume 88, 2016, Pages 330–335

7th Annual International Conference on Biologically Inspired Cognitive Architectures, BICA 2016

Computational Load Balancing Algorithm for Parallel Knapsack Packing Tree Traversal

Mikhail A. Kupriyashin[1] and Georgii I. Borzunov[1,2]

[1] National Research Nuclear University "MEPhI" (Moscow Engineering Physics Institute), Moscow, Russia
kmickle@yandex.ru

[2] Moscow State University of Design and Technology, Moscow, Russia
parproc@gmail.com

Abstract

The paper considers efficient computational load distribution for the exact parallel algorithm for the knapsack problem based on packing tree search. We propose an algorithm that provides for static and dynamic computational load balancing for the problem in question.

Keywords: knapsack problem, branch and bound, packing tree, load balancing, parallel computation

1 Introduction

Algorithms based on branch and bound approach are common among the exact knapsack problem algorithms. One of the ways to implement such an algorithm is to construct a packing tree. The algorithm then traverses the tree and all the branches known to yield no solution are cut (deleted) in advance[1, 2]. Drastic differences in the number of nodes in different branches is typical for these trees. When parallel implementation is in question, this feature causes difficulties with computational load balancing[3].

Load balancing efficiency may be improved provided there is a conversion between the ordinal number of the packing induring the traversal and its vector representation. This approach is referred to as linear arrangement[4] of the packing tree nodes. We suggest an algorithm to convert an ordinal number of a packing to the corresponding binary packing vector. Given this, we can treat all the packings as a sequence of numbered elements. This sequence may be split into equal pieces, thus providing near-equal load distribution.

 Selection and peer-review under responsibility of the Scientific Programme Committee of BICA 2016

doi:10.1016/j.procs.2016.07.444

2 Load Balancing based on Linear Arrangement of Packing Tree Nodes

Let us consider a knapsack problem with task size n being solved on a parallel computational platform with m processors. In order to provide static load balancing for the exact knapsack problem algorithm, based on packing tree search it is sufficient to split the sequence of packings into equal pieces and send these pieces to the worker processors as their subtasks. Each of the processors (with rank $0 \leq i < m$) uses the conversion algorithm to transform the ordinal number $(2^n/m) \cdot i$ into the corresponding packing vector and commits traversal starting with this vector. Thus the initial equal distribution of computational load is achieved (as long as processing a single packing vector is considered the base operation).

As some branches are cut during the tree traversal, the initial balance of workload distribution is lost. For instance, when the following task $\{\vec{a} = (10; 4; 3; 2; 1); w = 11\}$ is solved on a parallel platform with $m = 2$ processors (the corresponding graph is provided on Fig. 1, the first processor cuts the branches with the following roots: $(1; 1; 0; 0; 0)$;$(1; 0; 1; 0; 0)$;$(1; 0; 0; 1; 0)$. Its workload therefore reduces from 16 to 5 base operations (processing of a single packing vector). At the same time, no branch cuts occur for the second processor. So, the sequential algorithm would take 21 base operation, while the parallel implementation utilizing two processors takes 16 base operations, therefore the parallel computations efficiency value is $E = \frac{(21/16)}{2} \approx 66\%$. This proves the necessity to implement dynamic load balancing for the parallel algorithm to be highly efficient.

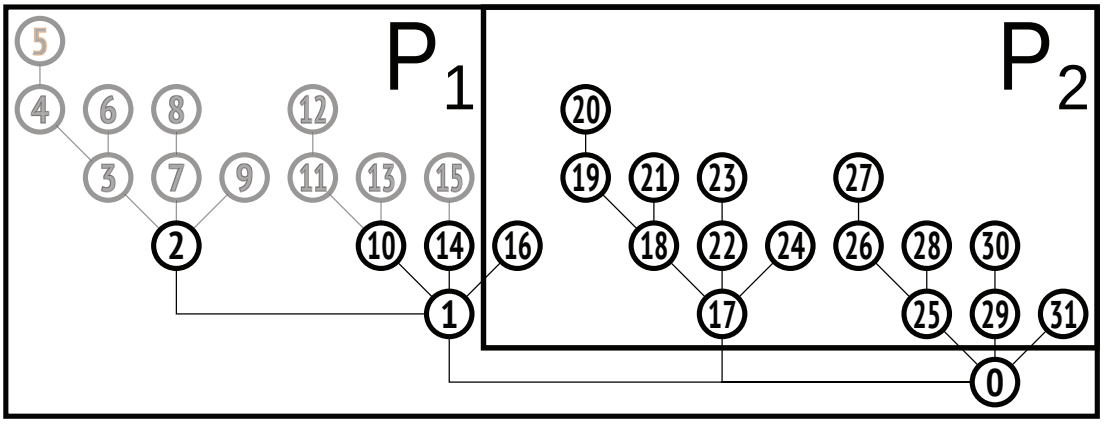

Figure 1: Example of computational load balance loss due to branch cuts (cut nodes are greyed out; P_1, P_2 — processor numbers).

In order to enable dynamic load balancing, all the processors should keep track of current amount of packing vectors to be processed (T_i). At start, this value equals to $(2^n/m)$ packings. Whenever a branch is cut, the processor is to reduce T_i by the amount of nodes in the cut branch. This amount depends on the count of trailing zeroes in the branch root vector. For instance, cut branch with the root packing vector being (0;1;0;0;0;0) has $2^4 - 1 = 15$ more nodes; root vector being (1;0;1;1;0) — $2^1 - 1 = 1$ node; being $(1; 1; 0; 0; 0; 0; 0; 0)$ — $2^6 - 1 = 63$ nodes. At a regular basis the master processor should gather the T_i values and reconsider the load balancing as necessary.

Parallel implementation of the packing tree search algorithm may have higher computational