Coalition Structure Formation using Parallel Dynamic Programming

Samriddhi Sarkar, Pratik Kumar Sinha, Narayan Changder and Animesh Dutta
Department of Computer Science and Engineering, National Institute of Technology Durgapur, India

Keywords: Multiagent System, Coalition Formation, Dynamic Programming, Parallelism.

Abstract: Dynamic Programming (DP) is an effective procedure to solve many combinatorial optimization problems and optimal Coalition Structure generation (CSG) is one of them. Optimal CSG is an important combinatorial optimization problem with significant real-life applications. The current best result in terms of worst case time complexity is $O(3^n)$. So, there is a need to find speedy approaches. This paper proposes a parallel dynamic programming algorithm for optimal CSG. We performed the comparison of our algorithm with the DP for CSG problem which happens to be a sequential procedure. The theoretical, as well as the empirical results show that our proposed method works faster than its sequential counterpart. We obtain a speed-up that is almost 14 times in case of 17 agents using a 16-core machine.

1 INTRODUCTION

Coalition formation is one of the pivotal coordination mechanisms in multi-agent system where a group of autonomous agents come forward and form alliances. Such alliances are needed when a task can not be accomplished by a single agent or a group of agents need to yield the result faster than that of a single agent (Rahwan et al., 2015). Coalition formation consists of 3 activities:

- Coalition value calculation.
- Partitioning the set of agents into the most beneficial one.
- Dividing the revenue earned among the agents.

This paper focuses on the second step that involves partitioning a set of agents into disjoint subsets so as to maximize the sum of the coalition values of the corresponding coalitions. Coalition formation has become ubiquitous in the past several years. In many real world problems, for example, e-commerce, where customers can form a group in case of bulk purchasing (Tsvetovat and Sycara, 2000). In distributed vehicle routing (Sandhlom and Lesser, 1997), companies form coalition to supply the deliveries. Another example of coalition formation in real life is the carpooling problem (Arib and Aknine, 2013), where users want to share their means of transportation. Social Ridesharing problem, where a set of commuters, connected through a social network, arrange one-time ride at short notice (Bistaffa et al., 2015). Surveillance of an area can be improved using autonomous sensors (Glinton et al., 2008).

To solve this CSG problem, researchers have explored different approaches. They can be typically clustered into 3 categories:

- Dynamic Programming (DP) algorithms: Preliminary work on this technique started way back in 1980's. Advantage of this algorithm is that, given a finite number of agents, it guarantees to find an optimal solution in $O(3^n)$ (Rahwan and Jennings, 2008). Dynamic Programming (DP) avoids repetition by storing values of the subproblems in memory. First dynamic programming algorithm for CSG was proposed in 1986 by Yun Yeh (Yeh, 1986). Improved version of DP (Rahwan and Jennings, 2008) proposed by Rahwan et al. performs fewer operation than DP.
- Anytime algorithm: With short execution time and strict deadlines, DP becomes futile as it needs to be run to completion. An anytime algorithm can return a valid solution even if it is interrupted before it ends. It works by generating an initial solution within a bound from the optimal, then gradually improve its quality as it search more search space. First anytime algorithm for CSG problem was proposed by (Sandholm et al., 1999) with worst-case guarantees. Such algorithm operates on the space between $\omega(n^n)$ to $O(n^n)$. (Dang and Jennings, 2004) proposed an anytime algorithm that produces solutions that are within a finite bound from the optimal. A few more algo-

Sarkar, S., Sinha, P., Changder, N. and Dutta, A.
Coalition Structure Formation using Parallel Dynamic Programming.
DOI: 10.5220/0006587401030110
In *Proceedings of the 10th International Conference on Agents and Artificial Intelligence (ICAART 2018) - Volume 2*, pages 103-110
ISBN: 978-989-758-275-2

rithms are (Rahwan et al., 2007), (Adams et al., 2010), (Changder et al., 2016).

- Heuristic: Objective of heuristic is to produce a *good enough* solution for a given problem within a reasonable amount of time. Due to the exponential nature of the problem, it is used when relatively quick result is needed. In the paper (Shehory and Kraus, 1998), authors have proposed an greedy algorithm, putting restriction on the size of the coalition.

The rest of the paper is organized as follows: Section 2 describes the background and relates work of optimal CSG problem. Section 3 describes the DP algorithm for the same. Section 4 focuses on the motivation for implementing parallelism. Proposed parallel DP has been described in section 5. Section 6 and 7 discusses about evaluation and performance respectively. Finally section 8 draws conclusion.

2 BACKGROUND AND RELATED WORK

Definition 1 *Given a set of agents* $A = \{a_1, a_2, \ldots, a_n\}$, *all the non-empty subsets are the possible coalitions, denoted as*

- *each coalition is a subset of set A, i.e.,* $C = \{a_i | i \in [1,n]\}$.
- *the number of possible coalition is* $| C | = 2^n - 1$.
- *all the subsets are pairwise disjoint* $C_i \cap C_j = \emptyset$.

e.g., if a set has 3 agents, then the possible coalitions are $\{a_1\}$, $\{a_2\}$, $\{a_3\}$, $\{a_1, a_2\}$, $\{a_1, a_3\}$, $\{a_2, a_3\}$, $\{a_1, a_2, a_3\}$.

Definition 2 *Coalition structure (CS) over A is a partitioning of the agents into different coalitions. Set of all coalition structure is denoted as* $\prod^A$.

- $CS = \{CS \in \prod^A \mid CS \subseteq C\}$
- $\cup_{i=1}^{s}(C_i) = A$ *where* $C_i \in CS$ *and s=size of CS.*

e.g., if a set A has 3 agents $\{a_1, a_2, a_3\}$ *then the coalition structure will be* $\{\{a_1\}, \{a_2, a_3\}\}$, $\{\{a_2\}, \{a_1, a_3\}\}$, $\{\{a_3\}, \{a_1, a_2\}\}$, $\{\{a_1, a_2, a_3\}\}$, $\{\{a_1\}, \{a_2\}, \{a_3\}\}$

Definition 3 *Characteristics function game G is defined by a pair (A, f), where A is a set of agents participating in the game* $f : 2^A \to \Re$. *In our coalition structure generation (CSG) problem this function assigns a real value* $v(C)$ *to each of the coalitions.*

- $f : C \to v(C)$ *where* $C \subseteq A$
- *value of any coalition structure will be sum of the coalition value of each coalition present in the CS.*
- *our objective is to find the CS whose value* $v(CS)$ *is maximum so as to maximize the social welfare.* $CS^\star = argmax_{CS \in \prod^A} v(CS)$

Time required to enumerate all the coalition structures in a brute force manner is $\Omega(n^{n/2})$ (Sandholm et al., 1999). The number of subsets increases exponentially as the number of agents increases linearly. The DP algorithm was used by Sandhlom et al. (Lehmann et al., 2006) to determine the solution of another combinatorial problem, winner determination problem in auction.

Another algorithm IP, proposed by Talal Rahwan (Rahwan et al., 2009), is faster than DP by several order of magnitude for some data distributions. *Improved Dynamic Programming* (IDP) algorithm (Rahwan and Jennings, 2008) is an improved version of DP. Authors have proved it to be faster and also it requires less memory but the drawback is that it takes $O(3^n)$ time in worst case. A modified version of IDP (Rahwan et al., 2012), uses goodness of both the algorithms IDP and IP.

Optimal Dynamic Programming (ODP) algorithm (Michalak et al., 2016) performs only one third of DP's operation by avoiding the redundant operations without losing the guarantee of finding the optimal solution. A hybrid version of ODP, called ODP-IP is also proposed in (Michalak et al., 2016) and authors have shown it is faster than other algorithms empirically.

In multi-agent system, there are often limits on the time to determine the solution of the problem. For example, a rescue operation is to be carried out by a pack of *n* agents. In such a case, instead of enumerating the whole search-space to find out the best coalition structure, initiation of the job is much more important. Researchers are continuously trying to improve the time taken by DP but all of these approaches are *sequential*. Whereas there is a scope to implement *parallelism* in the existing algorithms. First parallel DP algorithm for CSG was proposed in (Svensson et al., 2013). (Cruz et al., 2017) is about parallelisation of IDP using traditional CPU. With the increasing availability of parallel computing techniques, researchers are using more computational power into existing method to shorten the computation time substantially (Kumar et al., 1994).

In this paper we present a parallel approach of the DP algorithm using master-slave model. It also presents an evaluation of the same, both analytically and empirically.

3 DYNAMIC PROGRAMMING ALGORITHM

Coalition Structure Generation (CSG) problem has been made faster by solving it with the help of Dynamic Programming (DP). The procedure is explained in Algorithm 1.

DP algorithm for CSG works by maintaining two tables. *Partition table* Part(C) and *Value table* Val(C). Former one stores the partition that gives the highest value and later one stores the values of corresponding partitioning. In order to do so, the algorithm first evaluates all possible ways of splitting up the coalition C into two and checks whether it is beneficial to split C or not. The highest value as well as the partition (or no partition) are stored in table Val(C) and Part(C) respectively. For example $C = \{a_1, a_3\}$, in this case either they will work separately i.e., $\{\{a_1\},\{a_3\}\}$ or they will form a coalition, i.e., $\{a_1, a_3\}$. Now, $v[\{a_1, a_3\}]$=60 and $v[\{a_1\}] + v[\{a_3\}]$=55 so, here DP will store $\{a_1, a_3\}$ in Part(C) table and 60 in Val(C) table. Every splitting $(C, C/C')$ is evaluated as $v(C') + v(C/C')$. Table 1. shows how DP works for 4-agents. In each step

Algorithm 1: Dynamic Programming algorithm.

Input: Set of all possible non- empty subsets of n agents $(2^n - 1)$. The value of any coalition C is $v(C)$. If no $v(C)$ is specified then $v(C) = 0$
Output: Optimal coalition structure $CS^*(n)$

```
1:  for i = 1 to n do
2:      for C ⊆ A, where |C| = i do   ▷ A is set of n agents
3:          Val(C) ← v(C)
4:          Part(C) ← {C}
5:          for C' ⊂ C do   ▷ for every possible way of splitting C into two halves
6:              if Val(C') + Val(C \ C') > v(C) then
7:                  Val(C) ← Val(C') + Val(C/C')
8:                  Part(C) ← {C', C \ C'}
9:              end if
10:         end for
11:     end for
12: end for
13: CS* ← {A}
14: for C ∈ CS* do
15:     if Part(C) ≠ {C} then
16:         CS* ← (CS* \ C, Part(C))
17:         Go to line 14 and start with the new CS*
18:     end if
19: end for
20: Return CS*(n)
```

Table 1: Example of DP program with 4 agents.

Size	C	v(C)	All splitting by DP	Part(C)	Val(C)
1	$\{a_1\}$	30	$v[\{a_1\}] = 30$	$\{a_1\}$	30
	$\{a_2\}$	40	$v[\{a_2\}] = 40$	$\{a_2\}$	40
	$\{a_3\}$	25	$v[\{a_3\}] = 25$	$\{a_3\}$	25
	$\{a_4\}$	45	$v[\{a_4\}] = 45$	$\{a_4\}$	45
2	$\{a_1, a_2\}$	50	$v[\{a_1, a_2\}] = 50, v\{a_1\} + v\{a_2\} = 70$	$\{a_1\}\{a_2\}$	70
	$\{a_1, a_3\}$	60	$v[\{a_1, a_3\}] = 60, v\{a_1\} + v\{a_3\} = 55$	$\{a_1, a_3\}$	60
	$\{a_1, a_4\}$	80	$v[\{a_1, a_4\}] = 80, v\{a_1\} + v\{a_4\} = 75$	$\{a_1, a_4\}$	80
	$\{a_2, a_3\}$	55	$v[\{a_2, a_3\}] = 55, v\{a_2\} + v\{a_3\} = 65$	$\{a_2\}\{a_3\}$	65
	$\{a_2, a_4\}$	70	$v[\{a_2, a_4\}] = 70, v\{a_2\} + v\{a_4\} = 85$	$\{a_2\}\{a_4\}$	85
	$\{a_3, a_4\}$	80	$v[\{a_3, a_4\}] = 80, v\{a_3\} + v\{a_4\} = 70$	$\{a_3, a_4\}$	80
3	$\{a_1, a_2, a_3\}$	90	$v[\{a_1, a_2, a_3\}] = 90, v\{a_1\} + v\{a_2, a_3\} = 95$ $v\{a_2\} + v\{a_1, a_3\} = 100, v\{a_3\} + v\{a_1, a_2\} = 95$	$\{a_2\}\{a_1, a_3\}$	100
	$\{a_1, a_2, a_4\}$	120	$v[\{a_1, a_2, a_4\}] = 120, v\{a_1\} + v\{a_2, a_4\} = 115$ $v\{a_2\} + v\{a_1, a_4\} = 110, v\{a_4\} + v\{a_1, a_2\} = 115$	$\{a_1, a_2, a_4\}$	120
	$\{a_1, a_3, a_4\}$	100	$v[\{a_1, a_3, a_4\}] = 100, v\{a_1\} + v\{a_3, a_4\} = 110$ $v\{a_3\} + v\{a_1, a_4\} = 105, v\{a_4\} + v\{a_1, a_3\} = 105$	$\{a_1\}\{a_3, a_4\}$	110
	$\{a_2, a_3, a_4\}$	115	$v[\{a_2, a_3, a_4\}] = 115, v\{a_2\} + v\{a_3, a_4\} = 120$ $v\{a_3\} + v\{a_2, a_4\} = 110, v\{a_4\} + v\{a_2, a_3\} = 110$	$\{a_2\}\{a_3, a_4\}$	120
4	$\{a_1, a_2, a_3, a_4\}$	140	$v[\{a_1, a_2, a_3, a_4\}] = 140, v\{a_1\} + v\{a_2, a_3, a_4\} = 150$ $v\{a_2\} + v\{a_1, a_3, a_4\} = 150, v\{a_3\} + v\{a_1, a_2, a_4\} = 145$ $v\{a_4\} + v\{a_1, a_2, a_3\} = 145, v\{a_1, a_2\} + v\{a_3, a_4\} = 120$ $v\{a_1, a_3\} + v\{a_2, a_4\} = 145, v\{a_1, a_4\} + v\{a_2, a_3\} = 145$	$\{a_1\}\{a_2, a_3, a_4\}$	150

DP actually checks all the possible combinations i.e., $\binom{n}{s}$. Each combination then generates 2^{s-1} partitions. Running time of DP is $\binom{n}{s} \cdot O(2^k) = O(3^n)$.
To fill up the entries of Val(C) table of size-s, all the precedent coalition of size $1, 2, \ldots s-1$ are calculated and then those values are simply reused. Once the DP determines all the entries of both Part(C) and Val(C) tables, the optimal coalition structure CS^* is computed recursively. DP starts from the grand coalition $\{a_1, a_2, a_3, a_4\}$ and find highest valued coalition structure is $\{\{a_1\},\{a_2, a_3, a_4\}\}$. Next it looks for $\{a_2, a_3, a_4\}$ and finds beneficial split is $\{\{a_2\},\{a_3, a_4\}\}$ and lastly it finds a_3 and a_4 works together as shown in Table 1. As a result, DP concludes that the optimal solution is $\{\{a_1\},\{a_2\},\{a_3, a_4\}\}$

4 MOTIVATION FOR PARALLELISM

DP solves a given problem by dividing it into subproblems and later on solves those subproblems and eventually combines those results in order to obtain the final result. To compute the values Val(C) (Table. 1) of coalition size s, algorithm calculates Val(C) of all coalition of size $1, 2, \ldots, s-1$. Once all the entries of both the table Part(C) and Val(C) are computed, algorithm looks for the optimal solution $CS^\star$ recursively. A 3-size coalition $\{\{a_1, a_3, a_4\}\}$ is divided into $2^{s-1} = 2^2 = 4$

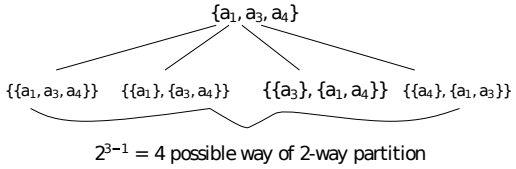

Figure 1: Independent tasks.

possible ways, i.e., $\{\{a_1,a_3,a_4\}\}$, $\{\{a_1\},\{a_3,a_4\}\}$, $\{\{a_3\},\{a_1,a_4\}\}$,$\{\{a_4\},\{a_1,a_3\}\}$. To compute step-1, DP looks into the memory where all the values are stored and to compute the values of the rest of the steps, it reuses values from previously stored entries.

Parallel algorithm design technique allows the designer to perform some of the tasks concurrently. Independent subproblems (subproblems with no data dependencies) (Stivala et al., 2010) are executed simultaneously and then all the individual results are summed up to generate the desired result. To get executed parallelly, an algorithm should possess some or all of the following traits (Kumar et al., 1994):

1. portions of the work that can be performed simultaneously.
2. mapping the concurrent pieces of work onto multiple processors running in parallel.
3. distribution of inputs, outputs to the process.

4.1 Decomposition

The process of dividing a computational task into smaller and independent subprocess, which can be executed parallely is called process/task decomposition. Here, in this coalition formation problem, two types of parallelism are possible:

1. The first type of parallelism can be achieved while Algorithm 1 performs exhaustive search for the highest valued 2-way partitioning of each s-size coalition i.e., computation of 2^{s-1} number of partitioning can be performed independently and concurrently. Figure 1 shows all possible 2-way partitioning of coalition $\{\{a_1,a_3,a_4\}\}$.
2. The second type of parallelism can be deployed in each step. CREW SM SIMD model allows concurrent-read (Aki, 1989)from the shared memory. It is also evident that all the s-size coalitions are independent of each other. All they need is to fetch the previously calculated values from table Val(C) of coalition size $1,2,3,\ldots,s-1$. Hence in each step $\binom{n}{s}$ number of coalition formation can be easily parallelised. Figure shows example of one such possibility.

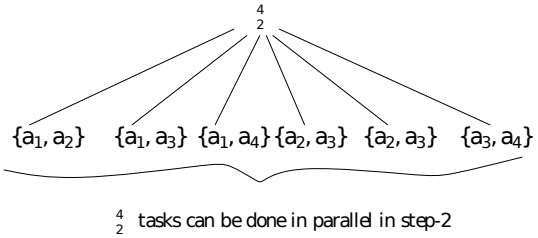

Figure 2: Independent tasks.

5 PARALLEL DYNAMIC PROGRAMMING

A CREW SM SIMD computer consists of p processors $P_1,P_2,\ldots,P_p$. Here we describe a parallel algorithm (Algorithm 2) for parallel computation of coalition structure generation (CSG) problem. For this, the algorithm will take (2^n-1) no. of $v(C)$s and produce optimal coalition structure CS^*. In order to design a parallel algorithm following properties (Aki, 1989) need to be satisfied:

- Number of processors
- speedup

5.1 Number of Processors

Theorem 1 *For a problem of size m, the number of processors required by a parallel algorithm, denoted by p is smaller than m and is a sublinear function of m, i.e., $p<m$ and $p=m^x$ where $0<x<1$.*

Proof 1 *Total no. of tasks in each s-sized coalition as described in section 3 is, $m=\binom{n}{s}(2^{s-1})$,*
Let no. of processors is even, i.e., $p=2b$, where $b\in N$

$$\sum\binom{n}{s}=2^n$$

total no. of terms=$n+1$
half of the terms = $\frac{(n+1+1)}{2}$
sum of half of the terms=$\frac{2^n}{2}$

$$= \quad 2^{n-1}$$

approximation of each term is

$$\frac{2^{n-1}}{(n+2)/2} \quad = \quad \frac{2^n}{n+2}$$

Hence, each term $\binom{n}{s}=\frac{2^n}{n+2}$
From the above theorem,

$$p \quad = \quad n^x$$

putting all the values

$$\begin{aligned} 2b &= (\frac{2^n}{n+2} \cdot 2^{s-1})^x \\ &= \frac{1}{(n+2)^x} \cdot (2^{n+s-1})^x \end{aligned}$$

again $(\frac{1}{n+2})^x$ *is always less than 1 or* 2^0.

So the number of processor will be,

$$\begin{aligned} 2b &= 2^0 \cdot (2^{n+s-1})^x \\ &= 2^{0+(n+s-1)\cdot x} \\ &= 2^{(n+s-1)\cdot x} \end{aligned}$$

5.2 Speedup

Theorem 2 *Running time of the parallel algorithm* $T(n)$ *(where n is size of the input) is smaller than the sequential algorithm for the problem at hand.*

Proof 2 *Total no. of processor our proposed system will use is* $2^{(n+s-1)\cdot x}$,
there are total $\binom{n}{s}$ *coalition in each step and each of them takes* (2^{s-1}) *time, so total amount of task (in terms of time) that can be done in parallel is* $m = \binom{n}{s}(2^{s-1})$,

$$\begin{aligned} m &= \binom{n}{s}(2^{s-1}) \\ &= \frac{2^n}{n+2} \cdot 2^{s-1} \\ &= \frac{1}{n+2} \cdot 2^{n+s-1} \end{aligned}$$

parallel execution on $p = 2b$ *no. of processors in each step will take* $t(n) = \frac{m}{p}$ *amount of time.*

$$\begin{aligned} \frac{m}{p} &= \binom{n}{s}\frac{2^{s-1}}{2b} \\ &= \binom{n}{s}\frac{2^{s-1}}{2^{(n+s-1)x}} \\ &= \frac{2^n}{n+2} \cdot \frac{2^{s-1}}{2^{(n+s-1)x}} \\ &= \frac{1}{n+2} \cdot 2^n \frac{2^{s-1}}{2^{(n+s-1)x}} \\ &= \frac{1}{n+2} \cdot \frac{2^{n+s-1}}{2^{(n+s-1)x}} \\ &= \frac{1}{n+2} \cdot 2^{(n+s-1)(1-x)} \end{aligned}$$

Now, each step will take $t(n)$ *time and calculation of step* $1,2,\ldots,n$ *will be done sequentially. Eventually our proposed algorithm will take*

$$\begin{aligned} T(n) &= \sum_{s=1}^{n} t(n) \\ &= \sum_{s=1}^{n} \frac{1}{n+2} \cdot 2^{(n+s-1)(1-x)} \\ &\approx n \cdot \frac{1}{n+2} \cdot 2^{(n+s-1)(1-x)} \\ &\approx 2^{(n+s-1)(1-x)} \end{aligned}$$

Upper bound of x is 1. If x tends to 1 then $(1-x)$ *tends to 0. And applying hit and trial method it is evident that*

$$\begin{aligned} \lim_{(1-x)\to 0}(n+s-1)(1-x) &< n \\ \Rightarrow 2^{\lim_{(1-x)\to 0}(n+s-1)(1-x)} &< 2^n \\ again\ 2^n &< 3^n \\ T(n) &< 3^n \\ \Rightarrow 3^n &> T(n) \\ \frac{3^n}{T(n)} &> 1 \end{aligned}$$

Time taken by DP is O(3^n*). Hence Speedup is greater than 1.*

5.3 Algorithm Details

Proposed Parallel Dynamic Programming (PDP) for CSG works the same way as DP does, which is described in Section 3. But the difference is that some of the computations are done in parallel manner in PDP. In traditional DP, in each step, $\binom{n}{s} \cdot 2^{s-1}$ number of calculations are done sequentially i.e., one by one, whereas in PDP, each step divides all possible coalition i.e., $\binom{n}{s}$ in chunks of slave processes. Each chunk contains at least 2 slave processes. Furthermore, all the coalitions inside a chunk divides them into 2^{s-1} parts.
Table 2. below shows which portion of the Algorithm 1 can be done concurrently. Three different colours denotes 3 such different portions. Blue row denotes that all the size-2 coalitions i.e., $\binom{4}{2}$=6, which are independent of each other. Likewise green denotes size-3 coalitions and lastly red denotes size-4 coalition for 4 agents. For example, in size-3, $\binom{4}{3}$=4 coalitions are divided into two chunks of slave processes (or sub processes) as described in Figure 3. $\{a_1,a_2,a_3\}$ and $\{a_1,a_2,a_4\}$ in slave process 1, similarly $\{a_1,a_3,a_4\}$ and $\{a_2,a_3,a_4\}$ in slave process 2. Now each slave process, in this example has $2 \times 2^{3-1} = 8$ computations to perform.

In this study, we have used up to 16 cores to implement a master-slave parallel dynamic programming (Li et al., 2013) to compute the optimal coalition

Algorithm 2: Parallel Dynamic Programming algorithm.

Input: Set of all possible non- empty subsets of n agents $(2^n - 1)$. The value of any coalition C is $v(C)$. If no $v(C)$ is specified then $v(C) = 0$
Output: Optimal coalition structure $CS^*(n)$

1: **for** $i = 1$ to n **do**
2: $C \subseteq A$, where $|C| = i$ ▷ A is set of n agents
3: $Val(C) \leftarrow v(C)$
4: $Part(C) \leftarrow \{C\}$
5: **end for**$s \in 2,\ldots,n$
6: ▷ //do it in parallel//
7: **for** $i \in 1,\ldots,ns$ **do**
8: $C \subseteq A :| C |= i$
9: $Val(C) \longleftarrow v(C)$
10: $Part(C) \longleftarrow C$
11: ▷ //do it in parallel//
12: **for** $i \in 1,\ldots,2^{s-1}$ **do**
13: each split of Subset C into two $C', C/C'$
14: ▷ if C is not Split then $C' = CandC/C' = 0$
15: ▷ C will be divided in every possible way into two halves
16: **if** $Val(C\prime) + Val(C/C\prime) > v(C)$ **then**
17: $Val(C) \longleftarrow Val(C\prime) + Val(C/C\prime)$
18: $Part(C) \longleftarrow (C\prime, C/C\prime)$
19: **end if**
20: **end for**
21: **end for**
22: $CS^* \longleftarrow Ag$
23: **for** every $C \in CS^*$ **do**
24: **if** $Part(C) \neq C$ **then**
25: $CS^* \longleftarrow (CS^*/C, Part(C))$
26: start with new CS^*
27: **end if**
28: **end for**
29: Return $CS^*(n)$

value. A *master process* in each step divides all possible coalitions into groups of *slave processes* depending on the number of the processors we have. Master-slave model (Li et al., 2014) is a commonly used paradigm for parallelizing a DP algorithm. Here the master process has a full version of DP algorithm and slave process has only the code for the subtasks. Slave processes evaluate their individual tasks and stores the result in the memory of the master process. Computation of all the slave processes are done in parallel, computation of the subtasks inside each subtask are done in sequential manner. Slave processes are responsible for carrying out the subtasks by determining the highest valued coalition structure (CS) of each coalition and send back it to their master process.

Table 2: Example of PDP program with 4 agents.

Size	C	v(C)	All splitting by DP	Part(C)	Val(C)
1	$\{a_1\}$	30	$v[\{a_1\}] = 30$	$\{a_1\}$	30
	$\{a_2\}$	40	$v[\{a_2\}] = 40$	$\{a_2\}$	40
	$\{a_3\}$	25	$v[\{a_3\}] = 25$	$\{a_3\}$	25
	$\{a_4\}$	45	$v[\{a_4\}] = 45$	$\{a_4\}$	45
2	$\{a_1,a_2\}$	50	$v[\{a_1,a_2\}] = 50, v\{a_1\} + v\{a_2\} = 70$	$\{a_1\}\{a_2\}$	70
	$\{a_1,a_3\}$	60	$v[\{a_1,a_3\}] = 60, v\{a_1\} + v\{a_3\} = 55$	$\{a_1,a_3\}$	60
	$\{a_1,a_4\}$	80	$v[\{a_1,a_4\}] = 80, v\{a_1\} + v\{a_4\} = 75$	$\{a_1,a_4\}$	80
	$\{a_2,a_3\}$	55	$v[\{a_2,a_3\}] = 55, v\{a_2\} + v\{a_3\} = 65$	$\{a_2\}\{a_3\}$	65
	$\{a_2,a_4\}$	70	$v[\{a_2,a_4\}] = 70, v\{a_2\} + v\{a_4\} = 85$	$\{a_2\}\{a_4\}$	85
	$\{a_3,a_4\}$	80	$v[\{a_3,a_4\}] = 80, v\{a_3\} + v\{a_4\} = 70$	$\{a_3,a_4\}$	80
3	$\{a_1,a_2,a_3\}$	90	$v[\{a_1,a_2,a_3\}] = 90, v\{a_1\} + v\{a_2,a_3\} = 95$ $v\{a_2\} + v\{a_1,a_3\} = 100, v\{a_3\} + v\{a_1,a_2\} = 95$	$\{a_2\}\{a_1,a_3\}$	100
	$\{a_1,a_2,a_4\}$	120	$v[\{a_1,a_2,a_4\}] = 120, v\{a_1\} + v\{a_2,a_4\} = 115$ $v\{a_2\} + v\{a_1,a_4\} = 110, v\{a_4\} + v\{a_1,a_2\} = 115$	$\{a_1,a_2,a_4\}$	120
	$\{a_1,a_3,a_4\}$	100	$v[\{a_1,a_3,a_4\}] = 100, v\{a_1\} + v\{a_3,a_4\} = 110$ $v\{a_3\} + v\{a_1,a_4\} = 105, v\{a_4\} + v\{a_1,a_3\} = 105$	$\{a_1\}\{a_3,a_4\}$	110
	$\{a_2,a_3,a_4\}$	115	$v[\{a_2,a_3,a_4\}] = 115, v\{a_2\} + v\{a_3,a_4\} = 120$ $v\{a_3\} + v\{a_2,a_4\} = 110, v\{a_4\} + v\{a_2,a_3\} = 110$	$\{a_2\}\{a_3,a_4\}$	120
4	$\{a_1,a_2,a_3,a_4\}$	140	$v[\{a_1,a_2,a_3,a_4\}] = 140, v\{a_1\} + v\{a_2,a_3,a_4\} = 150$ $v\{a_2\} + v\{a_1,a_3,a_4\} = 150, v\{a_3\} + v\{a_1,a_2,a_4\} = 145$ $v\{a_4\} + v\{a_1,a_2,a_3\} = 145, v\{a_1,a_2\} + v\{a_3,a_4\} = 120$ $v\{a_1,a_3\} + v\{a_2,a_4\} = 145, v\{a_1,a_4\} + v\{a_2,a_3\} = 145$	$\{a_1\}\{a_2,a_3,a_4\}$	150

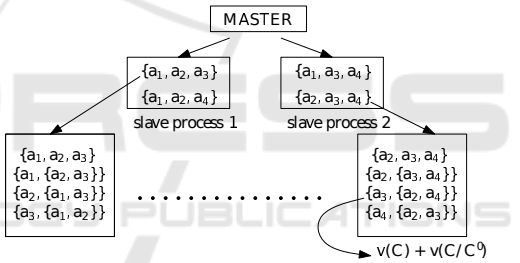

Figure 3: Illustration of master-slave parallel DP for CSG.

6 EVALUATION

This section first describes the experimental setup of our proposed algorithm.

6.1 Experimental Setup

The proposed Parallel DP algorithm for finding the optimal coalition structure (CS) is coded in Python. In order to provide a direct comparison with the serial version of the algorithm, the same is also coded in Python. The codes of the two algorithms were executed on a Dell PowerEdge R720 rack server with two eight core Intel Xeon E5-2600 processors and 96GB memory. Each execution was exclusive, i.e., no other user application or code was executed simultaneously. The inputs to the program, i.e., the values of the grand coalitions, was randomly generated using the Chi-square distribution ang Agent-based Normal distribu-

tion. The number of agents was varied from 10 to 17.

6.2 Dataset Generation

The NP-complete problems are intractable, i.e., they can be solved theoretically but in practice they take too long for their solutions to be useful. Coalition structure generation problem is also an NP-complete problem. We compared our proposed parallel dynamic programming algorithm with dynamic programming using following value distributions.

- **Chisquare** (χ^2)— The value of each coalition C is drawn from $v(C) \sim |C| \times \chi^2(\nu)$, where $\nu = 0.5$ is degrees of freedom.
- **Agent-based Normal**— as in (Michalak et al., 2016), each agent a_i is assigned a random power $p_i \sim N(10, 0.01)$. Then for all coalitions C in which agent a_i appears, the actual power of a_i in C is determined as $p_i^C \sim N(p_i, 0.01)$ and the coalition value is calculated as the sum of all the members' power in that coalition. That is, $\forall C, v(C) = \sum_{a_i \in C} p_i^C$.

7 PERFORMANCE

The parallel algorithm was executed with the same parameters as the serial algorithm. The algorithm computes the values of all coalition structures of a particular size in parallel. This is termed as completing one round of the execution. It was expected that the parallel algorithm would outperform the serial one. Through a theoretical analysis of the problem, the number of processors required, can be approximated and also we can achieve sub-linear speedup (Tan et al., 2007).

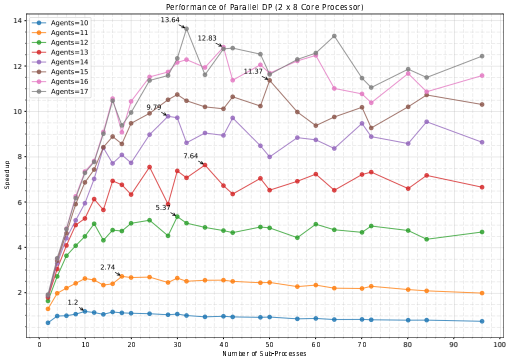

Figure 4: Performance analysis of DP and PDP to solve CSG problem for chi-square data distribution.

This grouping scheme was implemented and the maximum number of slave-processes (or sub-process) was kept fixed in a range between 2 to 96. The same hardware used in the previous evaluation tests was used to execute the modified algorithm. The graph shown in Figure 4 and Figure 5 represents the speed-up plots of varying number of agents against the number of slave-processes (sub-processes) for Chi-square and Agent-based Normal respectively. The arrows mark the highest speed-up for a particular number of agents, e.g., speed-up is almost 14 times in case of 17 agents.

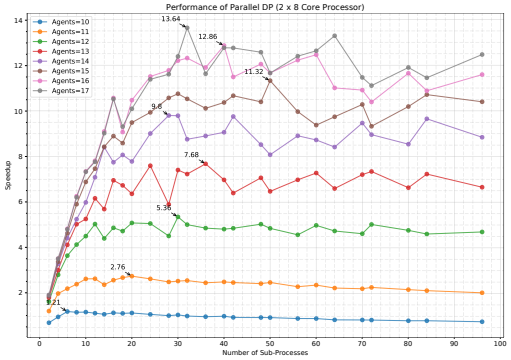

Figure 5: Performance analysis of DP and PDP to solve CSG problem for Agent-based Normal data distribution.

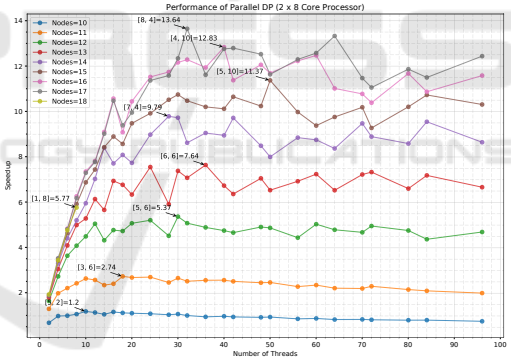

Figure 6: Performance analysis using various number of threads.

Figure 6 represents performance of our proposed algorithm using different number of threads. It has been observed that greater number of threads will not always result into greater speed-up.

8 CONCLUSION

Coalition structure generation is an NP-complete optimization problem in multi-agent system. This work presents a parallel dynamic programming for solving the CSG problem and has been compared with the traditional DP. Parallel dynamic programming is a well-known paradigm where dynamic programming meets

parallelism and results into a stronger method.

Evaluation shows our proposed technique is faster than traditional DP by several times, almost 14 times for 17 agents. Future work would implement the parallel DP algorithm for CSG problem using more parallel computing resources. Also we will focus on the relationship between the measure of parallelism (e.g., no. of processors), no. of agents and solution time.

ACKNOWLEDGEMENT

This research work is funded by DST (Department of Science and Technology), Govt. of India, with the research project grant No. - SB/FTP/ETA-0407/2013.

REFERENCES

Adams, J. A. et al. (2010). Anytime dynamic programming for coalition structure generation. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems.

Aki, S. G. (1989). *The design and analysis of parallel algorithms*. Old Tappan, NJ (USA); Prentice Hall Inc., USA.

Arib, S. and Aknine, S. (2013). Preferences and constraints for agent coalition formation. In *Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2013 IEEE/WIC/ACM International Joint Conferences on*. IEEE.

Bistaffa, F., Farinelli, A., and Ramchurn, S. D. (2015). Sharing rides with friends: A coalition formation algorithm for ridesharing. In *AAAI*.

Changder, N., Dutta, A., and Ghose, A. K. (2016). Coalition structure formation using anytime dynamic programming. In *International Conference on Principles and Practice of Multi-Agent Systems*. Springers.

Cruz, F., Espinosa, A., Moure, J. C., Cerquides, J., Rodriguez-Aguilar, J. A., Svensson, K., and Ramchurn, S. D. (2017). Coalition structure generation problems: optimization and parallelization of the idp algorithm in multicore systems.

Dang, V. D. and Jennings, N. R. (2004). Generating coalition structures with finite bound from the optimal guarantees. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*. IEEE Computer Society.

Glinton, R., Scerri, P., and Sycara, K. (2008). Agent-based sensor coalition formation. In *Information Fusion, 2008 11th International Conference on*. IEEE.

Kumar, V., Grama, A., Gupta, A., and Karypis, G. (1994). *Introduction to parallel computing: design and analysis of algorithms*. Benjamin/Cummings Redwood City, London, 2nd edition.

Lehmann, D., Müller, R., and Sandholm, T. (2006). The winner determination problem.

Li, X., Wei, J., Fu, X., Li, T., and Wang, G. (2013). Knowledge-based approach for reservoir system optimization.

Li, X., Wei, J., Li, T., Wang, G., and Yeh, W. W.-G. (2014). A parallel dynamic programming algorithm for multi-reservoir system optimization.

Michalak, T., Rahwan, T., Elkind, E., Wooldridge, M., and Jennings, N. R. (2016). A hybrid exact algorithm for complete set partitioning.

Rahwan, T. and Jennings, N. R. (2008). An improved dynamic programming algorithm for coalition structure generation. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*. International Foundation for Autonomous Agents and Multiagent Systems.

Rahwan, T., Michalak, T. P., and Jennings, N. R. (2012). A hybrid algorithm for coalition structure generation. In *AAAI*.

Rahwan, T., Michalak, T. P., Wooldridge, M., and Jennings, N. R. (2015). Coalition structure generation: A surveys.

Rahwan, T., Ramchurn, S. D., Dang, V. D., Giovannucci, A., and Jennings, N. R. (2007). Anytime optimal coalition structure generation. In *AAAI*, volume 7.

Rahwan, T., Ramchurn, S. D., Jennings, N. R., and Giovannucci, A. (2009). An anytime algorithm for optimal coalition structure generation.

Sandhlom, T. W. and Lesser, V. R. (1997). Coalitions among computationally bounded agents.

Sandholm, T., Larson, K., Andersson, M., Shehory, O., and Tohmé, F. (1999). Coalition structure generation with worst case guarantees.

Shehory, O. and Kraus, S. (1998). Methods for task allocation via agent coalition formation. volume 101, pages 165–200. Elsevier.

Stivala, A., Stuckey, P. J., de la Banda, M. G., Hermenegildo, M., and Wirth, A. (2010). Lock-free parallel dynamic programming.

Svensson, K., Ramchurn, S., Cruz, F., Rodriguez-Aguilar, J.-A., and Cerquides, J. (2013). Solving the coalition structure generation problem on a gpu.

Tan, G., Sun, N., and Gao, G. R. (2007). A parallel dynamic programming algorithm on a multi-core architecture. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*. ACM.

Tsvetovat, M. and Sycara, K. (2000). Customer coalitions in the electronic marketplace. In *Proceedings of the fourth international conference on Autonomous agents*. ACM.

Yeh, D. Y. (1986). A dynamic programming approach to the complete set partitioning problem.