Mapping Dynamic Programming Problems on Coarse Grained Multicomputer

Mounir Kechid
and Jean-frederic Myoupo
Picardie-Jules Verne University
MIS Laboratoiry, France.
Email: mounir.kechid@u-picardie.fr,
jean-frederic.myoupo(@)u-picardie.fr

Abdullah S.Khalufa
and Moteb S.Alghamdi
Computers Department
College of Engineering
Al-Baha University, KSA.
Email: abdullah.abdulaziz@live.com,
motebghamdi@gmail.com

***Abstract*—Parallelization of dynamic programming algorithms is a very well studied topic. In this paper we are interested by an important class of Dynamic Programming problems. Many results have been presented for this problems class for the fine grained parallel machines. We proposed an efficient parallel Coarse-Grained Multicomputer algorithm for a typical problem of this class. We present here the different possible execution scenarios of this algorithm and the implementation results of the most relevant one.**

I. Introduction

Parallel dynamic programming algorithm design is a very well studied topic. The goal is to study and define how to use parallel hardware for the processing of very large scale dynamic-programming-based solutions. Many parallel dynamic programming algorithms have been proposed, see e.g. [8],[9] for a survey. However, the majority of this algorithms suffer from two major inconveniences :

Efficiency-Portability Compromise
: There are either, *not portable* because they were derived for too realistic models as systolic and hypercube models where efficiency of solutions exclusively depends on the characteristics of these machines (interconnection networks...), or *non-implementable*, because too abstracte as PRAM (Parallel Random Access Machine) model. Thus, the design of efficient and portable solution is nowadays a challenge.

Software-Hardware Gap
: Most dynamic programming parallel algorithms are designed for fine-grained systems and shared memory machines. However, we witnessed the last decade a trend (migration) of parallel hardware to coarse-grained multiprocessor systems [17]. In the literature of parallel computing this incompatibility, between parallel algorithms and parallel hardware trend, is called:*"the software-hardware gap"* [18] or *"the parallel software barrier"* [5]. One of the mains challenges for researchers in parallel algorithms design is to reduce this gap [18]. One takes great interest to design efficient parallel algorithms for coarse-grained multiprocessors.

The most judicious solution for *Efficiency-Portability Compromise* is to be interested in the same way to the efficiency and the portability of the parallel algorithms. This requires a centre model which abstracts the details from low level of the parallel machines (to increase the portability), without misusing at the point to lead the designers to develop inefficient algorithms, by hiding to them crucial characteristics of the machines. It is accordingly that Valiant ([20], [21]) proposed the famous BSP model (Bulk synchronous parallel model).

The BSP model was refined by Dehne et al [6] to CGM (Coarse Grained Multicomputer) which is fully coarse-grained[1] version of BSP. Experimental results show that CGM algorithms map well to standard parallel hardware (Clusters) and exhibit speedups similar to what was predicted in their analysis [4].

In [14], we have proposed a CGM-based parallel algorithm for all the dynamic programming problems of the class presented below (paragraph II-A).
In this paper we present the different possible execution scenarios of this algorithm and the implementation results of the most relevant one.
The rest of the paper is organized as follows. Section II presents peleminaries about the problem studied class and the BSP/CGM algorithms. Section III presents the solution of the studied problem as calculations of shortest-paths (one to all) in a weighted multi-levels DAG. A decomposition of this DAG is detailed in section IV. In section V two CGM-execution scenarios are presented. The task scheduling methods are described in the section VI. The section VI presents and analyzes the implementation results. A conclusion ends this paper.

[1] coarse-grained on calculations and communications

II. PRELIMINARIES

A. Studied problems class

Dynamic programming is a technique which can be applied to solve several combinatorial optimization problems. It is used for a wide range of applications. In this paper we are interested in a typical dynamic programming problems class. It's the class of all combinatorial optimization problems that can be modelled by (1). The table I shows some important problems of this class and their application fields.

$$Cost(i,j) = \begin{cases} Init(i) & 1 \leq i = j \leq n. \\ \min\limits_{i \leq k < j} \{Cost(i,k) + Cost(k+1,j) \\ \qquad +F(i,k,j)\} & 1 \leq i < j \leq n. \end{cases} \quad (1)$$

In equation (1), n is the size of the problem, and the values $Init(i)$ and $F(i,j,k)$ are known or can be easily computed. The optimal solution of the problem at hand is given by $Cost(1,n)$. In the function F, $F(i,k,j)$ gives for the sub-problem (i,j), the junction cost of the two optimal solutions of (i,k) and $(k+1,j)$ in order to produce optimal solution for (i,j). This function, called *union function*, depends in the intrinsic characteristics of concerned problem.

The number of possible solutions for a problem of size n, is exponential in n: $O(4^n/n^{3/2})$. Thus, a solution based on the direct exhaustive search is very poor. In the early 1970s, several researchers noted the presence of characteristics, like *optimality* and the *overlapping sub-problems*, in the *Matrix Chain Ordering Problem* (MCOP for short). Applying the technique of dynamic programming, in 1973, Godbole [10] proposed the first polynomial time solution for the MCOP. It requires $O(n^3)$ time steps on $O(n^2)$ memory size. The structure and the complexity of this solution being independent of the MCOP *union* function F, it has become the standard algorithm for solving all problems that can be modelled by (1). This is why this algorithm is often said *generic algorithm* in the literature. Its general structure is given in Algorithm 1.

Figure 1 shows the form of the n-levels DAG (with $n = 4$). As a matter of fact, the Algorithm 1 solves sub-problems, corresponding to the multi-levels DAG, using a bottom-up approach. Level by level from the lowest level. It is called "*dynamic programming algorithm*". For each treated sub-problem the value of its optimal solution is saved in a table ((n,n) triangular matrix) called "*dynamic programming table*".

Problem	Application fields
Matrix chain ordering problem	Scientific computing
Optimal binary search tree problem	Information processing
Optimal convex polygon triangulation problem	Computational geometry
Optimal objects parenthetisation problem	Industrial processing

Table I: Some important problems of the studied class.

Algorithm 1: The general structure of Godbole algorithm

```
1  for i = 1 to n do
2  |  Cost(i,i) ← Init(i) ;
3  end
4  for d = 2 to n do
5  |  for i = 1 to n − d + 1 do
6  |  |  Cost(i, i + d − 1) ← ∞;
7  |  |  for k = i to i + d − 2 do
8  |  |  |  Temp ← Cost(i,k) + Cost(k + 1, i + d −
   |  |  |  1) + F(i,k,i + d − 1);
9  |  |  |  if Temp< Cost(i,i+d-1) then
10 |  |  |  |  Cost(i, i + d − 1) ← Temp ;
11 |  |  |  end
12 |  |  end
13 |  end
14 end
```

The calculation of $Cost(1,n)$ involves the resolutions of all sub-problems (i,j) / $1 \leq i \leq j \leq n$. The study of dependency between sub-problems shows that they are organizable on a *n-levels Directed Acyclic Graph* (*DAG* for short) where

- Each node represents a sub-problem. A level d is a set of nodes representing the sub-problems set $\{(i, i+d-1)/ 1 \leq i \leq n-d+1\}$.
- An arc from a node representing (i,j) to the one representing (i',j') means: computing of $Cost(i',j')$ depends on $Cost(i,j)$.
- The node with no outgoing arc represents the original problem to solve, $Cost(1,n)$.

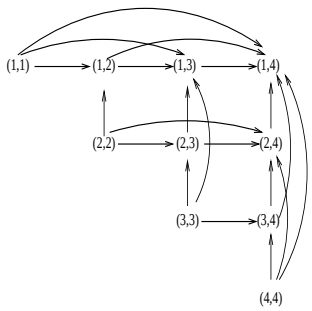

Figure 1: 4-levels DAG schematizing dependencies between different sub-problems involved in the calculation of $Cost(1,4)$

B. BSP/CGM Algorithms

In the BSP/CGM model a parallel machine is a set of p processors, each having an own local memory of size M $(M > p)$ and connected to a router able to deliver messages in point-to-point fashion[2]. A CGM algorithm consists of alternating local computation and global communication rounds. In each communication round each processor communicate at most $O(M/p)$ data units. M being the memory space requirement of the sequential algorithm used in the parallelization process. The design of efficient CGM algorithm has two main goals : (1) find a good processors load balancing in most of the computation rounds, and (2) minimize the number of communication rounds, ideally independent from the problem size.

III. Shortest Paths in Dynamic Graph

The dynamic programming problems are often solved through the shortest paths problems on weighted DAGs. For the problems class described in II-A, a graphic model, called *dynamic graph*, appeared in [1]. It is noted D_n for a problem of size n. The Figure 2 shows the dynamic graph D_n for $n = 4$. It has the same form of DAG representing the dependency between subproblem presented above, with an additional node $(0, 0)$. It is shown in [1] that the calculation of $Cost(i, j)$ is equivalent to search in D_n the shortest path from node $(0, 0)$ to (i, j). Denoting by $SP(i, j)$ the length of this shortest path, the dynamic graph D_n is defined by

1. Set of nodes (vertices)
$S = \{(i, j) : 1 \leq i \leq j \leq n\} \cup \{(0, 0)\}$

2. Set of edges which is the union of two edges subsets:
Unit edges: $\{(i, j) \longrightarrow (i, j+1) : 1 \leq i \leq j < n\} \cup \{(i, j) \uparrow (i-1, j) : 1 < i \leq j \leq n\} \cup \{(0, 0) \nearrow (i, i) : 1 \leq i \leq n\}$
Jumps: $\{(i, j) \Longrightarrow (i, t) : 1 < i \leq j < t-1 < n\} \cup \{(s, t) \Uparrow (i, t) : 1 \leq i < s-1 < t \leq n\}$

3. Weight function w where

$$
\begin{aligned}
&w((0,0) \nearrow (i,i)) = 0 \\
&\qquad if \quad 1 \leq i \leq n \\
&w((i,j) \rightarrow (i,j+1)) = F(i,j,j+1) \\
&\qquad if \quad 1 \leq i \leq j < n \\
&w((i,j) \uparrow (i-1,j)) = F(i-1,i-1,j) \\
&\qquad if \quad 1 < i \leq j \leq n \\
&w((i,j) \Longrightarrow (i,t)) = SP(j+1,t) + F(i,j,t) \\
&\qquad if \quad 1 < i \leq j < t-1 < n \\
&w((s,t) \Uparrow (i,t)) = SP(i,s-1) + F(i,s-1,t) \\
&\qquad if \quad 1 \leq i < s-1 < t \leq n
\end{aligned}
$$

Therefore it's straightforward to proof that Godbole algorithm presented above (algorithm 1) is equivalent to perform the calculation of the shortest paths from $(0, 0)$ to others vertices in a dynamic graph D_n. Diagonal after diagonal, from left to right. A table called *Shortest Paths Matrix*[3], $SP(n, n)$, is used to save in the cell $SP(i, j)$ the shortest path from node $(0, 0)$ to (i, j), $1 \leq i \leq j < n$.

[2]Regardless of the means used (shared memory, networks, hybrid architectures ...). This ensures the generality of the model, i.e. many parallel architectures can be instances of the abstract machine defined by the BSP / CGM.

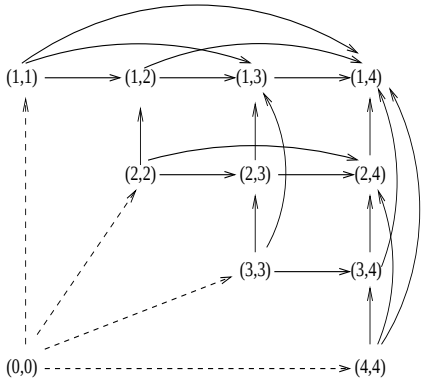

Figure 2: Dynamic graph D_4

Given a problem of size n and its corresponding dynamic graph D_n, it was shown in [1] that:
If the shortest path from $(0, 0)$ to (i, k) contains the jump $(i, j) \Rightarrow (i, k)$, then there exists a dual shortest path with the same cost containing the jump $(j + 1, k) \Uparrow (i, k)$.
This is fundamental for our BSP/CGM algorithm. It helps to avoid computation redundancy of the shortest paths costs in D_n.

IV. Data structures partitions

This section presents data structures involved in the construction of the shortest paths (one-to-all) in the weighted graph D_n, on a Coarse Grained Multicomputer of p processors, each having a local memory of size $O(n^2/p)$. To reach this goal, we describe the decomposition of the problem in sub-problems in section IV-A. We study the dependencies between sub-problems in section IV-B.

A. Decomposition of the problem in sub-problems

We decompose the graph D_n (hence the matrix $SP(n, n)$) in $p(p + 1)/2$ sub-graphs (submatrix of SP) as shown in Figure 3. Each subgraph of D_n correspond to a submatrix of SP. Hereafter, We will use indifferently the term *"block"* to denote a subgraph or its corresponding submatrix. To localize blocks, We'll use the same matrix terminology. By SP_{ij}, We'll denote the block (i,j). Let SP_{ij} and SP_{km} be two blocks of the, decomposition of D_n.

***Definition** 1:* SP_{ij} and SP_{km} are

- on the same line i if $i = k$
- on the same column j if $j = m$
- on the same diagonal $j - i + 1$ if $j - i = m - k$

***Claims** 1:* The number of blocks is :

- $p - i + 1$ in line i
- j in column j

[3]The equivalent of dynamic programming table.