**International Conference on Automation, Control, Engineering and Computer Science (ACECS'14)
Proceedings - Copyright IPCO-2014, pp.19-27
ISSN 2356-5608**

On Greedy Algorithms for Solving the Matrix Chain Product Problem with Square Dense and Triangular Matrices

Khaoula Bezzina[#1], Zaher Mahjoub[#2]

[#] *Faculty of Sciences of Tunis, University of Tunis El Manar*
University Campus - 2092 Manar II, Tunis, Tunisia

[1]khaoulabezzina@gmail.com

[2]Zaher.Mahjoub@fst.rnu.tn

***Abstract*— We address a combinatorial optimization problem (COP), namely a particular variant of the matrix chain product problem (MCPP) where the chain involves square dense and triangular matrices. Besides the classical dynamic programming algorithm of cubic complexity, exact greedy algorithms (GA) of linear complexity are also known in the literature. We present in this paper new exact GA's of linear complexity too and leading in general to different optimal chain parenthesizations (OCP). Our objective is to optimize, while building an OCP represented by a binary tree, its parallelism degree. Indeed, this criterion is quite useful for an efficient parallel computation of the chain product according to a given OCP. Our theoretical contribution is validated by an experimental study permitting to establish comparisons between several OCP's through their representative binary trees, thus precising the practical interest of the approaches we developed.**

***Keywords*— Binary tree, complexity, dynamic programming, grain, greedy algorithm, matrix chain product, parallel algorithm, parenthesization.**

I. Introduction

The matrix chain product problem (MCPP) is an easy COP arising in various scientific applications e.g. in robotics, computer animation, process control, mathematical programming, etc ... [1][2].

Given a chain of, say, n matrices A_1, …, A_n where the dimensions of A_i (i=1…n) are (p_{i-1},p_i), the (standard) MCPP consists in constructing an optimal chain parenthesization (OCP) with minimal number of operations (MNO) for computing the product matrix $A_1…A_n$. An exact and optimal dynamic programming algorithm (DPA) of complexity $O(n^3)$ is known in the literature since 1973 [3]. A more recent algorithm based on polygon triangulation (PT) of complexity O(nlogn) is also known [4].

A particular variant of the MCPP in which we are interested, denoted MCPP-DLU, corresponds to the case where the chain matrices are square and either dense (D), lower (L) or upper triangular (U). Besides the DPA, two exact greedy algorithms (GA) of complexity O(n) are known since 2011[5].

It has to be underlined that an OCP may be represented by a binary tree (RBT) where each node corresponds to a matrix product and the root to the chain product. Since the MCPP-DLU may admit, in general, several solutions i.e. OCP's whose corresponding RBT's may be differently structured, an interesting and worthwhile aspect of the problem may be raised here. It consists in constructing different OCP's and define criteria permitting to establish an accurate comparison between them, hence choose the best according to these criteria. It turns out that the structural properties of the corresponding RBT's may well be used for this purpose.

The main objective of our contribution based on [5] consists in fact in the design of new greedy algorithms leading to OCP's optimizing the structure of the corresponding RBT's. To be more precise, this optimization targets an efficient parallel computation of the matrix chain product. For this purpose, starting from two known GA's called GA and AGA, denoted henceforth GA1 and GA2, we first propose two modified versions denoted GA1' and GA2'. We then propose two new algorithms denoted GA3 and GA4. The rationale behind these series of algorithms is to generate an OCP, thus an RBT leading to the fastest parallel computation of the chain product.

The remainder of the paper is organized as follows. In section 2, after giving some useful definitions on OCP's and RBT's, we proceed to a description of the two known greedy algorithms GA1 and GA2, then our four new greedy algorithms GA1', GA2', GA3 and GA4. Section 3 is devoted to an experimental study permitting to establish a comparison between the diverse algorithms previously discussed and validate our contribution. Finally, we conclude our work and propose some perspectives in section 4.

International Conference on Automation, Control, Engineering and Computer Science (ACECS'14)
Proceedings - Copyright IPCO-2014, pp.19-27
ISSN 2356-5608

II. GREEDY ALGORITHMS

A. *Representative Binary Tree of an Optimal Chain Parenthesization*

As previously precised, an optimal (or any) chain parenthesization of a matrix chain product may be represented by a binary tree (RBT) [4]. This latter is built such that the leaves are in the top level and the root at the bottom. We precise that each node corresponds to a matrix product, the leaves correspond to the product of two successive matrices of the chain i.e. (A_i,A_{i+1}) and the root to the final product. Furthermore, each node is weighted by the amount of computation (i.e. number of operations) required by the associated matrix product.

If we adopt a level decomposition of the RBT where each level involves independent i.e. non related nodes, we can define for the RBT a height (or a depth) h and a width w. The height h is defined as the number of levels in the decomposition i.e. the maximal number of nodes in a path relating a leaf to the root. The width corresponds to the maximal number of nodes in a level. Remark that the RBT is in fact a top-down directed graph [6].

Both height and width are important characteristics of a RBT representing a given OCP and may be used as criteria in order to establish a fine comparison between different OCP's through their associated RBT's. We'll see further a third criterion leading to a finer comparison. Let us add that for a given RBT, we may define in general several level decompositions having the same height but different widths [6].

On the other hand, the height may be seen as the number of successive steps required to perform the chain product, whereas the width may be considered as the number of independent substeps involved by each step. Height and width may be easily bounded as follows [5]:

$$\lceil \log_2 n \rceil \leq h \leq n-1 \;\; ; \;\; 1 \leq w \leq \lfloor n/2 \rfloor$$

The height upper bound (n-1) and width lower bound (1) are both reached for a chain-structured RBT and mean that each step involves only one substep. Two particular associated OCP's are the standard left-right parenthesization (LRP) i.e. $((\ldots((A_1A_2)A_3)\ldots)A_{n-1})A_n)$ and its symmetric, the right-left parenthesization (RLP) i.e. $(A_1(A_2(\ldots(A_{n-2}(A_{n-1}A_n))\ldots)$. On the other hand, the height lower bound ($\lceil \log_2 n \rceil$) and width upper bound ($\lfloor n/2 \rfloor$) correspond, particularly when n is a power of 2, to a complete binary tree (CBT). The corresponding parenthesization is induced by the so called *Associative Fan In Algorithm* (AFIA) whose basic principle is to combine the chain matrices couple by couple. One may think that a CBT is an optimal RBT i.e. the best suited for the parallel computation of the chain since the maximal number of matrix couples is performed in parallel. Although this case corresponds to the maximal parallelism degree, it scarcely corresponds to an OCP and may induce a higher amount of computations (see the experimental study further). In fact, one very decisive criterion is as detailed below.

Let us consider a path relating a leaf to a root in the RBT. Clearly, it involves h nodes. Such a path is called *critical* if the sum of the costs (or weights) of its h nodes is maximal, the cost (or weight) of a node being the number of operations required to perform the corresponding matrix product. Using the standard terminology in parallel computing, the RBT is called a task graph where each task corresponds to a node and has a cost (the cost or weight of the node). It is well known that, disposing of an unlimited number of (identical) processors, the optimal (i.e. minimal) time to execute the task graph, i.e. compute in parallel the matrix products of the RBT, is equal to the cost of a critical path (ccp) [5][6][7]. Remark here that a number of processors equal to the width w of the RBT is sufficient to reach this minimal time [6][8]. Therefore, given different OCP's and their associated RBT's, we'll use for their comparison, in addition to the heights and widths, the discriminating ccp criterion.

B. *Improved Greedy algorithms GA1' and GA2'*

Consider a chain of n square matrices $A_1,\ldots, A_n$ of size, say p, where each matrix may be either dense (■:D), lower triangular (◣:L) or upper triangular (◥:U).

Let c(AB) be the cost (number of operations) required for computing two square matrices of size p where A and B are either D, L or U. Concerning both structures and costs, we recall that :

- ■■=■;■◣=■;◣■=■;■◥=■;◥■=■;◣◥=■; ◥◣=■;◣◣=◣;◥◥ =◥
- $c(DD)=2p^3+O(p^2)$; $c(DU)=c(UD)=c(DL)=c(LD)=p^3+O(p^2)$
- $c(LU)=c(UL)=2p^3/3+O(p^2)$; $c(LL)=c(UU)=p^3/3+O(p^2)$

$$\begin{aligned}\text{Therefore } c(DD) &= 2c(DU) = 2c(UD) = 2c(DL) = 2c(LD)\\ &= 3c(UL) = 3c(LU)\\ &= 6c(LL) = 6c(UU)\end{aligned}$$

Remark that (i) the highest (resp. lowest) cost occurs when both A and B are dense (resp. triangular of same structure) and (ii) combining an L matrix with a U one creates a D one. These relations are in fact at the basis of the two greedy algorithms introduced in [5]. Their basic idea consists in avoiding, whenever it is possible, to increase the number of D-matrices, since a new D-matrix is created when combining an L one with a U one.

The greedy algorithms, denoted GA1 and GA2, involve in fact two phases as follows:

- A compressing phase consisting in computing every sub-chain involving matrices of same structure. The LRP (→) or the RLP (←) are used here. This leads to a compressed chain (C_C) where any two successive matrices are necessarily of different structures.
- Compute the C_C by avoiding the creation of new D matrices.

Remark that the authors considered several cases in this second phase and precised for each a procedure leading to a final OCP.

The following example illustrates GA1 and GA2:

n=15, C = DDDDDUDDLUUDLDL, MNO= (21+1/3) p^3 (we restrict MNO and ccp to p^3 terms for sake of simplicity)

- GA1: final result : (h,w,ccp) = (12,3,19)
  - Phase 1: C = ((((DD)D)D)D)U(DD)L(UU)DLDL
    → C_C = DUDLUDLDL
  - Phase 2: C_C = (((((((DU)D)L)U)D)L)D)L
- GA2 : final result : (h,w,ccp) = (8,5,15)
  - Phase 1: C_C = DUDLUDLDL (same as in GA1)
  - Phase 2: C_C = (((DU)((DL)U)))(DL))(DL)

We can see that GA2 improves GA1 (lower h, larger w and lower ccp).

We underline the fact that for a subchain involving matrices of same structure, any parenthesization is optimal i.e. leads to the same number of operations (NOP). Therefore, we may choose anyone. Our first contribution consisted in improving both GA1 and GA2 by designing two new algorithms, denoted GA1' and GA2', where we used, in the two phases, the Associative Fan-In algorithm (AFIA) when combining matrices of same structure in order to increase the parallelism degree of the final OCP.

Remark. As previously precised AFIA combines matrices couple by couple. Let m be the length of the processed chain. When $m=2^q$, there is only one chain parenthesization (CP). For instance, for m=8, we get the following CP:
$((((A_1A_2)(A_3A_4))((A_5A_6)(A_7A_8))))$.

But, when m is not a power of 2, we may define several CP's. For instance for m=7, we have the following CP's:

$((A_1A_2)(A_3A_4))((A_5A_6)A_7)$; $((A_1A_2)(A_3A_4))(A_5(A_6A_7))$
$((A_1A_2)A_3)((A_4A_5)(A_6A_7))$; $(A_1(A_2A_3))((A_4A_5)(A_6A_7))$ …

Therefore, when applying AFIA several times on successive subchains whose lengths are not powers of 2, we must choose judicious CP's. It could be done by defining in this case a completed chain whose length is $m^*=2^q$ where $q = \lceil \log_2 n \rceil$, the m^*- m created matrices being of size 0.

We can now describe algorithms GA1' and GA2' as follows (Fig. 1) by adopting the same notations as in [5]:

GA1'	**GA2'**
Phase1	
Compression phase: compute the product of every subchain involving matrices of same structure by using AFIA. Let C_C be the obtained compressed chain.	
Phase2	
Case 1. C_C involves no D matrix: compute C_C according to either LRP (→) or RLP (←)	
Case 2. C_C involves at least one D matrix. C_C may be written as follows: $C_C=C_1DC_2$ where C_1 involves no D-matrix but C_2 may involve. ❖ $C_1=\emptyset$: compute DC_2 according to LRP. ❖ $C_2=\emptyset$: compute C_1D according to RLP. ❖ $C_1\neq\emptyset$, $C_2\neq\emptyset$: Compute C_1D according to RLP. Then, the result being a D-matrix, compute DC_2 according to LRP. An alternative consists in computing DC_2 according to LRP. Then, the result being a D-matrix, compute C_1D according to RLP.	***Case 2.*** C_C involves at least one D-matrix. C_C may be written as follows: $C_C=C_1DC_2D…C_iD…C_{r-1}DC_r$ where each subchain C_i, i=1…r, involves no D matrix. ❖ $C_1=C_r=\emptyset$ i.e. $C_C=DC_2DC_3…C_iD…C_{r-1}D$: compute DC_i, i=2…r-1, according to LRP. Then compute the resulting chain involving r-1 D-matrices according to AFIA ❖ $C_1=\emptyset$, $C_r\neq\emptyset$ i.e. $C_c=DC_2DC_3…C_iD…C_{r-1}DC_r$: compute DC_i, i=2…r, according to LRP. Then, compute the resulting chain involving r-1 D-matrices according to AFIA. ❖ $C_1\neq\emptyset$, $C_r=\emptyset$ i.e. $C_c=C_1DC_2DC_3…C_iD…C_{r-1}D$: compute C_iD, i=1…r-1, according to RLP. Then, compute the resulting chain involving r-1 D-matrices according to AFIA. ❖ $C_1\neq\emptyset$, $C_r\neq\emptyset$ i.e. $C_c=C_1DC_2DC_3…C_iD…C_{r-1}DC_r$: compute C_iD, i=1…r-1, according to RLP. Then, (i) compute the resulting chain involving r-1 D-matrices according to AFIA; (ii) compute DC_r according to LRP. Or symmetrically compute DC_i, i=2…r, according to LRP. Then (i) compute the resulting chain involving r-1 D-matrices according to AFIA; (ii) compute (C_1D) according to RLP.

Fig. 1 Greedy Algorithms GA1' & GA2'

As for GA1 and GA2, GA1' and GA2' are obviously of linear complexity i.e. O(n).

We reconsider the previous example:

n=15, C = DDDDDUDDLUUDLDL, MNO= (21+1/3) p^3

- GA1': final result : (h,w,ccp) = (11,4,17)
  - Phase 1: C = (((DD)(DD))D)U(DD)L(UU)DLDL
    $\rightarrow$ C_C = DUDLUDLDL
  - Phase 2: C_C = (((((((DU)D)L)U)D)L)D)L
- GA2': final result : (h,w,ccp) = (6,6,11)
  - Phase 1: C_C = DUDLUDLDL (same as in GA1')
  - Phase 2: C_C = ((DU)((DL)U))((DL)(DL))

This leads to the following RBT's (Fig. 2). We can see that GA1' improves GA1 (lower ccp) and GA2' improves GA2 (lower ccp).

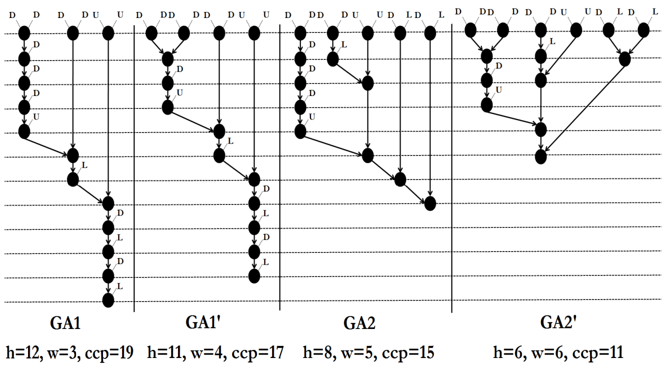


Fig. 2 RBT's for GA1, GA1', GA2 and GA2'

C. New Greedy Algorithms GA3 and GA4

1) GA3: It involves 3 phases. The first is, as for GA1' and GA2', a compressing phase. Remark here that combining D-matrices is not necessary and is postponed in GA3. Let C_C be the compressed chain. In the second phase, two cases have to be considered. The first occurs when C_C involves no D-matrix. Here we can use either LRP or a RLP. We may also mix the two. The second case occurs when C_C involves at least one D-matrix. The idea consists here in successively combining lowest cost couples as much as possible i.e. we begin by couples LD, DL, UD, DU until only D-matrices remain. Then we use AFIA to combine the remaining D-matrices.

GA3 is detailed below.

- **Phase1** : LU Compressing phase

Compute every subchain involving matrices of same structure **for only L and U** (D's are not considered) according to AFIA. The compressed chain may be written as follows:

$C_C = C_1D^*C_2D^*\ldots C_iD^*\ldots C_{r-1}D^*C_r$, where each subchain C_i, i=1…r involves no D-matrix and D^* is a subchain of D-matrices.

- **Phase 2** : Two cases have to be considered :

(i) Case 1. C_C involves no D-matrix: compute C_C according to either LRP or RLP. The two may also be combined.

(ii) Case 2. C_C involves at least one D-matrix. Proceed as follows.

- Step1. $C_1 \neq \emptyset$: compute C_1D according to RLP.
- Step2. $C_r \neq \emptyset$: compute DC_r according to LRP.

C_C may now be written a follows :

$C_C = D^*C_2D^*\ldots C_iD^*\ldots C_{r-1}D^*$. Each subchain C_i may be split into two subchains C_{i1} and C_{i2} of nearly same lengths.

- Step3. For each subchain $D^*C_{i1}C_{i2}D^*$, compute DC_{i1} according to LRP and $C_{i2}D$ according to RLP.

- **Phase 3**: Compute the resulting chain involving only D-matrices according to AFIA.

We reconsider the previous example:

n=15, C = DDDDDUDDLUUDLDL, MNO= (21+1/3) p^3 (MNO and ccp are given in terms of p^3 for sake of simplicity).

Final result : (h,w,ccp) = (6,6,9+1/3)

- Phase 1: C = DDDDDUDDL(UU)DLDL
  $\rightarrow$ C_C = DDDDDUDDLUDLDL
- Phase 2:
  - Step1 : C_C = DDDDDUDDLUDLDL (no modification)
  - Step2 : C_C = DDDDDUDDLUDL(DL)
    $\rightarrow$ C_C = DDDDDUDDLUDLD
  - Step3 : C_C = DDDD(DU)D(DL)(UD)(LD)
- Phase 3: C_C = (((DD)(DD))((DD)(DD)))D

The OCP is C = (((DD)(DD))(((DU)D)((DL)((UU)D))))(L(DL))

Notice that GA3 improves GA2' (lower ccp).

Remarks.

- Complexity. Phase 1 may be done in linear time by scanning the initial chain and saving the cutting indices that will used by AFIA. Phase 2 requires scanning the compressed chain in order to detect the subchains involving only D-matrices and process the C_iD, DC_i subchains. A linear time is also required here. Phase 3 requires scanning the obtained chain in order to find the cutting indices that will be used by AFIA. Thus, an overall linear time is required.

- Optimality. The proof is based on the two lemmas presented in [5]. According to the latter, in an OCP, matrices of same type (L or U) are necessarily combined together (Phase 1 in GA3). Moreover, if the compressed chain C_C involves no D-matrix, both LRP and RLP are optimal (Phase 2-Case 1). When C_C

involves at least one D-matrix, an OCP consists in never combining an L-matrix with a U-matrix (Phase 2-Case 2).

- When the input chain involves no D-matrix, it is computed, after been compressed, according to either LRP or RLP. The two ways lead in general to different OCP's.

2) GA4: It involves like GA3 three phases. The first and third are the same as in GA3. In fact, only Step 3 of Case 2 in Phase 2 is different.

We detail GA4 below.

- **Phase1** : LU Compressing phase *same as in GA3*
- **Phase 2** : Two cases have to be considered :
  (i) Case 1. Same as in GA3.
  (ii) Case 2. C_C involves at least one D-matrix. Proceed as follows.
    - Step1. $C_l \neq \emptyset$: compute C_lD according to RLP.
    - Step2. $C_r \neq \emptyset$: compute DC_r according to LRP.
    C_C may now be written a follows:
    $C_C = D^*C_2D^* \ldots C_iD^* \ldots C_{r-1}D^*$. Each subchain C_i may be split into two subchains C_{i1} and C_{i2} of nearly same lengths.
    - Step3. For each subchain $D^*C_{i1}C_{i2}D^*$, compute DC_{i1} according to LRP and $C_{i2}D$ according to RLP. If the subchains D^* involve more than 3 D-matrices (i.e. $D^*=DD^{\#}D$), then compute its inner subchain ($D^{\#}$) according to AFIA.
- **Phase 3** : Same as in GA3.

Remark that GA4 is clearly of linear complexity.

We use the previous example to illustrate the approach:
n=15, C = DDDDDUDDLUUDLDL, MNO= (21+1/3) p^3 (MNO and ccp are given in terms of p^3 for sake of simplicity).

Final result : (h,w,ccp) = (5,6,8)

- Phase 1: C_C = DDDDDUDDLUDLDL
- Phase 2:
  -Step1 : C_C = DDDDDUDDLUDLDL (no modification)
  -Step2 : C_C = DDDDDUDDLUDL(DL)
  $\rightarrow$ C_C = DDDDDUDDLUDLD
  -Step3 : C_C = (DD)(DD)(DU)D(DL)(UD)(LD)
- Phase 3: C_C = ((DD)(DD))((DD)D)

The OCP is C = (((DD)(DD))((DU)D))((((DL)(UU)D))(L(DL)))

The induced RBT's for GA2', GA3 and GA4 are as follows.

We can see from Fig. 3 that GA3 improves GA2' and GA4 improves GA3 (lower ccp).

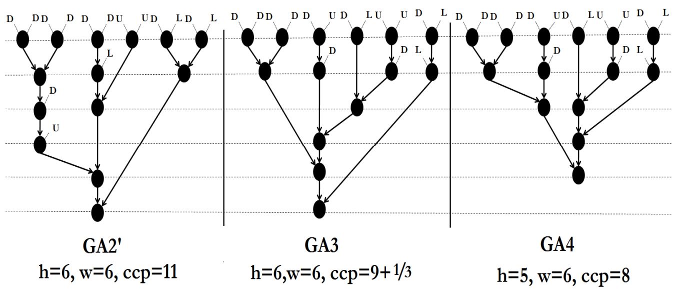


Fig. 3 RBT's for GA2', GA3 and GA4

A comparative table including the 6 GA's, LRP, RLP and AFIA is given below.

TABLE I
CHARACTERISTICS OF OCP'S AND NON OCP'S

Algorithm	MNO	h	w	ccp
GA1	21+1/3	12	3	19
GA2	21+1/3	8	5	15
GA1'	21+1/3	11	4	17
GA2'	21+1/3	6	6	11
GA3	21+1/3	6	6	9+1/3
GA4	21+1/3	5	6	**8**
LRP	NOP=22	14	1	22
RLP	NOP=22	14	1	22
AFIA	NOP=22+2/3	**4**	**7**	**8**

We can see that GA4 as well as AFIA gave the best ccp. However, AFIA is not optimal since it does not correspond to an OCP and requires $4p^3/3$ more operations than GA4.

III. COMPARATIVE EXPERIMENTAL STUDY

We achieved a series of experimentations in order to establish an accurate performance evaluation of the four designed algorithms. Intra and inter-algorithm comparative studies were also done. For this purpose we used the following parameters:

- MNO, h, w, ccp (previously defined)
- Intra-algorithm speed-ups :
  - S_1=MNO/ccp_GA1 ; $S_{1'}$=MNO/ccp_GA1'
  - S_2=MNO/ccp_GA2 ; $S_{2'}$=MNO/ccp_GA2'
  - S_3=MNO/ccp_GA3 ; S_4 =MNO/ccp_GA4
- Inter-algorithm versions speed-ups :
  - $S_{11'}$= ccp_GA1/ccp_GA1' ; $S_{22'}$= ccp_GA2/ccp_GA2'
- Inter-algorithm speed-ups :
  - $S_{1'2'}$= ccp_GA1'/ccp_GA2'; $S_{1'3}$=ccp_GA1'/ccp_GA3
  - $S_{1'4}$ = ccp_GA1'/ccp_GA4 ; $S_{2'3}$= ccp_GA2'/ccp_GA3
  - $S_{2'4}$ = ccp_GA2'/ccp_GA4 ; $S_{3\,4}$ = ccp_GA3 /ccp_GA4
- Speed-ups with regard to AFIA :
  - S_{3A}= ccp_AFIA/ccp_GA3 ; S_{4A}= ccp_AFIA/ccp_GA4

International Conference on Automation, Control, Engineering and Computer Science (ACECS'14)
Proceedings - Copyright IPCO-2014, pp.19-27
ISSN 2356-5608

TABLE II
MNO (GA'S), NOP (AFIA) AND RBT STRUCTURE

n	**MNO**	**GA1**		**GA1'**		**GA2**		**GA2'**		**GA3**		**GA4**		**AFIA**		
		h	**w**	**h**	**w**	**h**	**w**	**h**	**w**	**h**	**w**	**h**	**w**	**NOP**	**h**	**w**
50	188	35	11	35	12	19	17	12	18	9	17	8	18	240	6	25
75	251	50	18	50	19	15	27	10	28	9	29	10	31	337	7	37
100	361	71	24	71	25	27	37	11	38	10	38	9	39	459	7	50
125	430	83	26	83	30	27	37	20	41	14	45	12	45	552	7	62
150	551	95	37	95	43	36	51	12	57	11	52	11	56	707	8	75
175	637	130	36	130	38	50	62	14	64	11	65	10	66	817	8	87
200	638	136	47	136	51	42	66	13	70	11	72	11	72	845	8	100
225	800	151	47	151	56	60	72	14	81	15	81	12	81	1010	8	112
250	890	171	54	171	58	55	79	19	83	13	86	11	87	1132	8	125
275	960	184	61	181	69	65	88	16	96	13	101	11	101	1247	9	137
300	1099	202	68	202	76	65	89	14	97	13	109	12	111	1427	9	150
325	1159	216	70	214	83	76	102	18	115	13	116	11	116	1470	9	162
350	1308	239	79	239	86	86	110	14	117	13	124	11	126	1630	9	175
375	1310	253	88	253	98	88	125	14	135	13	143	12	143	1673	9	187
400	1412	260	85	257	103	89	123	16	141	13	146	13	148	1826	9	200

TABLE III
MNO , NOP AND CCP

n	**MNO**	**ccp**						**AFIA**	
		GA1	**GA1'**	**GA2**	**GA2'**	**GA3**	**GA4**	**NOP**	**ccp**
50	188	135	134	96	54	40	**36**	240	**36**
75	251	192	192	87	**39**	**39**	48	337	42
100	361	288	288	156	48	46	**42**	459	**42**
125	430	324	324	156	72	*57*	59	552	**42**
150	551	381	381	207	57	**48**	**48**	707	**48**
175	637	528	528	282	60	**48**	**48**	817	**48**
200	638	525	525	243	57	49	*53*	845	**48**
225	800	607	607	334	58	64	*54*	1010	**48**
250	890	672	672	327	81	60	*54*	1132	**48**
275	960	744	726	387	64	58	**54**	1247	**54**
300	1099	792	792	381	66	58	**54**	1427	**54**
325	1159	870	858	450	81	60	**54**	1470	**54**
350	1308	963	963	504	66	61	**54**	1630	**54**
375	1310	1014	1014	516	63	56	**54**	1673	**54**
400	1412	1014	1014	504	72	63	60	1826	**54**

We precise that we choosed 90 values for n in the range [5 450] and 3 input chains for each n. The input chains were randomly generated. The above tables depict excerpts of the results we obtained. MNO (for the GA's) and NOP (number of operations for AFIA) are given in terms of $p^3/3$ for sake of simplicity.

The analysis of the above results leads to the following remarks:

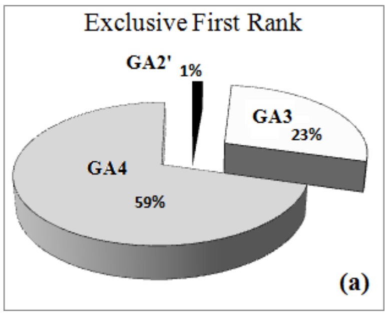
- The height of the RBT in GA1' (resp.GA2') is always lower or equal than the height in GA1 (resp. GA2).
- The width of the RBT in GA1' (resp.GA2') is always larger than the width in GA1 (resp. GA2).
- The width of the RBT in GA3 and GA4 is always larger than the width in GA2'.
- The largest width is given by AFIA.

The series of tests we achieved (excerpts are depicted in Tables II and III) permit to rank the different algorithms according to the induced ccp's (see Fig. 4). We have the following:

- GA4 (resp. GA3) gave the lowest ccp in 75% (resp. 39%) of the cases and was exclusively the best in 59% (resp. 23%) of the cases.
- GA2' gave the lowest ccp in 2% of the cases and was exclusively the best in 1% of the cases.

We have to add that AFIA gave the lowest ccp in 86% of the cases and was exclusively the best in 36% of the cases. However, it never induced an OCP.

Intra-algorithm speed-ups (based on Tables II and III and depicted in Fig. 5 and Table IV below) illustrate the previous remarks.



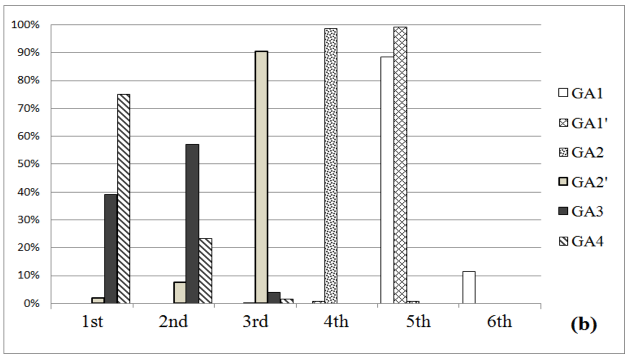


Fig. 4 Ranking of the GA's according to ccp

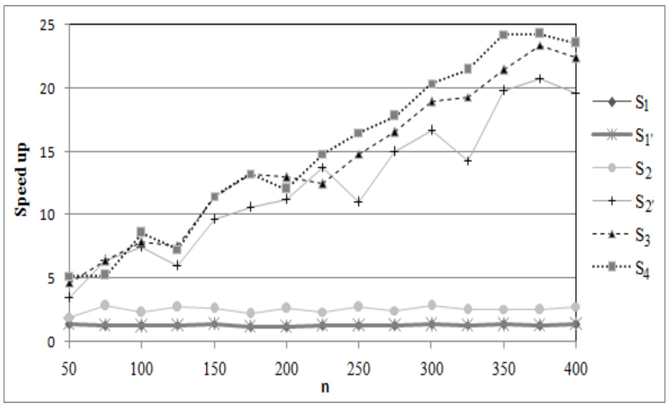


Fig. 5 Intra-algorithm speed-ups

TABLE IV
INTRA-ALGORITHM SPEED-UP VARIATION INTERVAL

Speed-up / Variation Interval	S_1	$S_{1'}$	S_2	$S_{2'}$	S_3	S_4
Lower bound	1.21	1.21	1.96	3.48	5.22	6.44
Upper bound	1.39	1.4	2.89	20.79	23.39	24.26

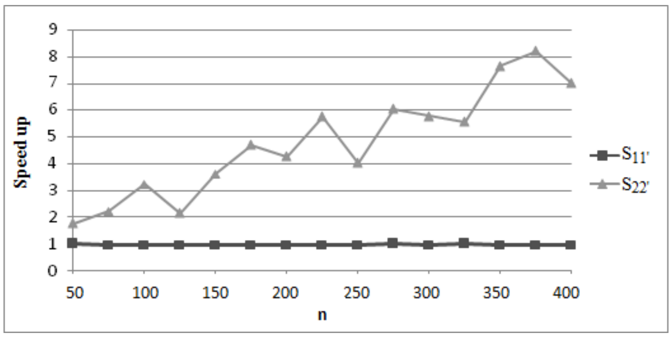


Fig. 6 Inter-algorithm versions speed-ups

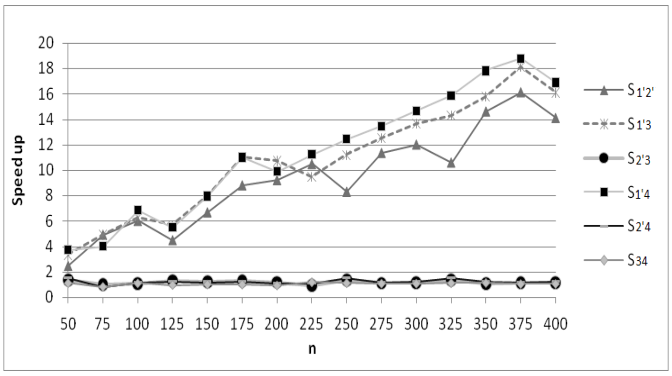


Fig. 7 Inter-algorithm speed-ups

TABLE V
INTER-ALGORITHM SPEED-UP VARIATION INTERVAL

Speed-up / Variation Interval	$S_{1'2'}$	$S_{1'3}$	$S_{2'3}$	$S_{1'4}$	$S_{2'4}$	S_{34}
Lower bound	2.48	3.35	0.91	3.72	0.81	0.81
Upper bound	16.1	18.11	1.35	18.78	1.5	1.19

From Fig. 6, we can remark that *parallel* GA2' is always faster (until 8 times) than *parallel* GA2. We hence deduce the improvement introduced relatively to both GA1 and GA2.

As to intra-algorithms speed-ups, we particularly notice that *parallel* GA3 and *parallel* GA4 are always better than the others (see Fig. 7 and Table V). To be more exhaustive, we have:

- *Parallel* GA2' is always better than *parallel* GA1' since $S_{1'2'}$ belongs to the interval [2.48 16.1].
- *Parallel* GA4 is almost always better than *parallel* GA2' since $S_{2'4}$ is larger than 1 in 99% of the cases (see Fig. 7).
- *Parallel GA3* is closely behind parallel GA4 i.e. S_{34} belongs to [0.81 1.19].

If we compare GA3 and GA4 with AFIA which corresponds to the more intuitive parallel algorithm for computing the matrix chain, we can see that the two former gave ccp's very close to AFIA ccp (see Fig. 8). However, AFIA never gave an OCP and required 29% to 70% more (sequential) operations than the others. For this reason, AFIA is to be rejected.

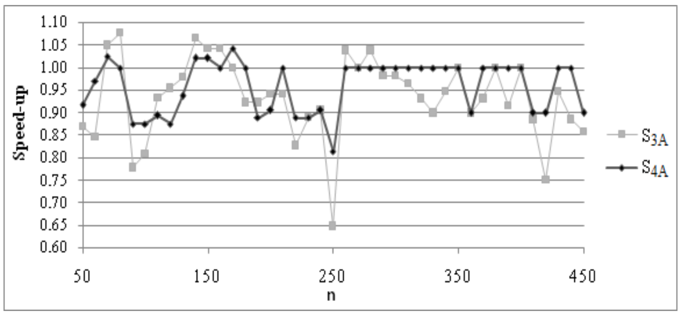


Fig. 8 Speed-ups GA3 and GA4 vs AFIA

We can therefore conclude that the experimental study confirmed the interest of GA3 and GA4 vs GA1 and GA2. Needless to say that none is always the best.

IV. Conclusion

In this paper, we studied a particular variant of the matrix chain product problem, namely a chain involving square dense and triangular matrices. We proposed new linear complexity greedy algorithms suited for an efficient parallel computation of the chain matrix according to an optimal chain parenthesization. The results we obtained enable us to precise the following interesting points that deserve to be studied:

- Given an OCP and the corresponding RBT and its tuple (h,w,ccp), determine the least number of processors p_{min} ($\leq$w) permitting the parallel computation of the chain matrix in optimal time i.e. equal to ccp.
- Define and study the impact of other discriminating criteria when comparing OCP's inducing the same ccp.
- Achieve a series of experimentations on a target parallel computer (e.g. multicore machine) in order to evaluate the practical performances of the designed algorithms.

References

[1] S. Ezouaoui, F. Ben Charrada, and Z. Mahjoub, "On instances of the matrix chain product problem solved in linear time", in *Proc. ROADEF 09*, Nancy, France, 2009.

[2] H. Lee, J. Kim, and S. Lee, "Evaluation of matrix chain products on parallel systems", in *Proc. PDCS'97*, Louisiana, U.S.A, 1997.

[3] S. S. Godbole, "An efficient computation of matrix chain products", IEEE Trans. Computers, vol. 22(9), pp. 864-866, 1973.

[4] H. Lee, J. Kim, S. Je Hong, and S. Lee, "Processor allocation and task scheduling of matrix chain products on parallel systems, *IEEE Trans. on Parallel and Distributed Systems*, vol. 14, pp. 394-407, 2003.

[5] F. Ben Charrada, S. Ezouaoui, and Z. Mahjoub, "Greedy algorithms for optimal computing of matrix chain products involving square dense and triangular matrices, *RAIRO-OP*, vol. 45, pp. 1-15, 2011.

[6] M. Cosnard, and D. Trystram, *Algorithmes et Architectures Parallèles*, InterEditions, 1993.

[7] F. Karoui-Sahtout, "Contribution à la conception d'un analyseur statique de programmes parallèles", M. Eng. thesis, Faculty of Science of Tunis, Tunis, Tunisia, 1996.

[8] Z. Mahjoub, and F. Karoui-Sahtout (1994). "Maximal and optimal degrees of parallelism for a parallel algorithm", in *Proc. Transputers'94*, Arc et Senans, France, 1994.