Correspondant au flot $\bar{f}'$ dans $G'_p(p)$, pour n'importe quel $x_{it}, x_{js} \in Y_p(p)$, on définit les temps

$$h'_{ij} = \begin{cases} h_{ij}, & \text{si } (J_{it}, x_{js}) \in A'_p(p) \text{ et } f_{ijt} < c'_{ijt}, \\ -h_{ji}, & \text{si } (x_{js}, x_{it}) \in A'_p(p) \text{ et } f_{jit} > 0, \\ \pm\infty, & \text{dans les autres cas.} \end{cases} \tag{9}$$

Ces temps définissent la matrice des temps sommets-sommets $H'_p(p)$. De même, définissons la matrice sommets-sommets $\bar{H}'_p(p)$ des temps pour n'importe quel $x_{it}, x_{js} \in \bar{Y}_p(p)$. En (9) nous prenons $h'_{ij} = +\infty$ ou $h'_{ij} = -\infty$, selon que nous cherchons les circuits négatifs respectivement positifs.

L'algorithme pour déterminer un flot de la catégorie des flots dynamiques maximaux avec les départs au plus tôt et les arrivées les plus tardives a les étapes suivantes.

Etape 1. On détermine un flot dynamique maximal $\bar{f}$ pour p périodes du temps et la coupe minimale avec l'algorithme présenté dans [4]. On construit $G'_p(p)$ avec le flot $\bar{f}'$. On passe à l'étape 2.

Etape 2. Pas 1. Correspondant au flot $\bar{f}'$ courant, on détermine la matrice $H'_p(p)$.

Pas 2. A l'aide de la matrice $H'_p(p)$ on cherche les circuits négatifs. S'il existe un tel circuit, on fait le changement de flot comme dans [7] et on passe au Pas 1. Sinon, on passe à l'étape 3.

Etape 3. On applique les pas de l'étape 2 concernant la matrice $\bar{H}'_p(p)$ et avec la spécification qu'on cherche les circuits positifs. S'il n'existe pas de circuits positifs, alors $\bar{f}'$ est le flot cherché.

Théorème 1. *L'algorithme exposé ci-dessus détermine un flot dynamique maximal avec les départs au plus tôt et les arrivées le plus tardives en un nombre fini d'itérations.*

Démonstration. L'affirmation de ce théoreme résulte du fait qu'un flot dynamique maximal avec les départs au plus tôt et les arrivées les plus tardives est un flot dynamique maximal du temps minimal aux départs et du temps maximal aux arrivées.

BIBLIOGRAPHIE

1. Berge, C. — *Graphes et hypergraphes*, Dunod, Paris, 1970.
2. Ciurea, E — *Deux remarques sur le flot dynamique maximal de coût minimal*, R.A.I.R.O., volume 13, Paris, 1979, 303—306.
3. Ciurea, E. — *Sur les flots dynamiques*, An. St. Univ. „Al. I. Cuza", Iași, tom. XXV, s-I-a, 1979, 73—77.
4. Ciurea, E. — *Les problèmes des flots dynamiques*, Cahiers du C.E.R.O., volume 26, no 1—2, Bruxelles, 1984, 3—9.
5. Ford, L R. and Fulkerson, D. R. — *Flow in Networks*, Princeton I University Press Princeton, N. J., 1962.
6. Gale, D. — *Transient Flows in Networks*, Michigan Math. J. 6, 1959, 59—63.
7. Klein, M. — *A Primal Method for Minimal Cost Flows*, Manag. Sci. 14, 1967, 205—220.

Reçu le 13. III. 1985

Université de Brașov
Roumanie

Analele științifice ale Universității „Al. I. Cuza" din Iași
Tomul XXXI, s. I a, Matematică, 1985, f. 3.

SYSTOLIC ALGORITHMS TO SOLVE LINEAR SYSTEMS BY ITERATION METHODS

BY

OCTAV BRUDARU

1. Introduction. The notion of systolic computation was introduced by Kung and Leiserson. The computation is described using a network of processors which act on flows of data, passing through the system. The processors compute rhythmically and all operands used in computation arrive at a processor simultaneously. At every tick the data can pass from a processor to its neighbours according to the communication graph and free global communication is disallowed. A systolic device perform its computation on behalf of a host.

Foster and Kung [2] consider that a systolic algorithm must be designed so that it can be implemented by means of only a few types of simple processors, the data and control flow of the algorithm is simple allowing cells to be connected by a network with local and regular interconnections and the algorithm extensively employs pipelining and parallel computation.

The time to solve just one instance of a problem is the response time while the period of system [1] is the time elapsed between the solutions of two consecutive problems inputs to a system.

Many examples of systolic algorithms are given in [4] and [5]. In the sequel we shall focus on systolic algorithms to solve a linear system, $Ax=b$.

Leiserson presents in [4] a linearly connected array to solve $Ax=b$, where A is a nonsingular $n\times n$ lower triangular matrix with the band width $w=q$. The solution is obtained in $2n+q$ clock ticks (CT).

Another way to solve $Ax=b$ is to use the LU-decomposition. In [4] it is given a hex-connected systolic array for pipelining the LU-decomposition assuming no pivoting is needed. The computation is executed in $3n+$ $+\min(p, q)$, where A is an $n\times n$ band matrix with band width $w=p+q-1$. After the factorization, the solving of linear systems $Ly=b$ and $Ux=b$ takes $4n+p+q$ CT and two distinct networks are used.

Kramer presents in [3] a systolic algorithm to invert an $n\times n$ nonsingular matrix in $4n$ CT with a two-dimensional systolic array using Gauss's algorithm and assuming no pivoting is needed. If pivoting is required then an $O(n^2)$-time systolic algorithm is given.

In the next section we present a systolic algorithm to solve a linear system by the Gauss-Seidel method. The design of a systolic array able to implement the acceleration method is discussed in Section 3. The proposed algorithms satisfy the design requirements above mentioned.

2. A systolic variant of the Gauss-Seidel method. A classical iteration method is the Gauss-Seidel method, which converges if for *e.g.* A is an irreductible diagonally dominant matrix. Given an arbitrary starting approximate solution $x^{(0)}$, if A is an $n\times n$ matrix then an interation is defined as

(1) $$z_i=\Sigma_{j=1}^{i-1}a_{ij}x_j^{(m+1)},$$

(2) $$y_i=\Sigma_{i=+1}^{n}a_{ij}x_j^{(m)},$$

(3) $$x_i^{(m+1)}=(b_i-z_i-y_i)/a_{ii},$$

for $i=1,\dots,n$ and $m\geq 1$. Suppose that A is a band matrix with band width $w=p+q-1$. We shall use the inner-product-step processor ($IPSP$) and the division processor (DP) described in Figure 1.

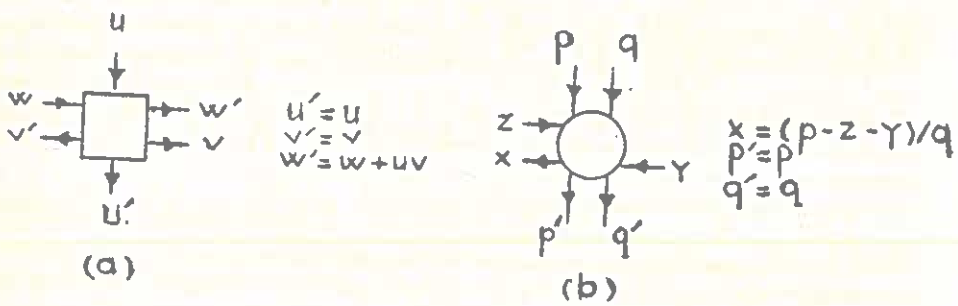

Figure 1. The I.P.S.P. *(a)* and the D.P. *(b)*.

The proposed systolic algorithm is based on a two-dimensional systolic array which is constructed using the linear systolic array (LSA). The LSA is shown in Figure 2.

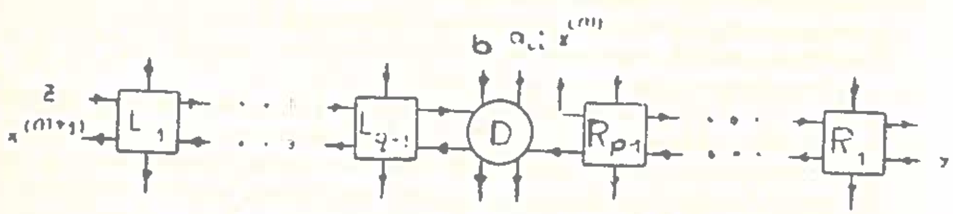

Figure 2. The L.S.A that executes one iteration.

This LSA is designed to accomplish just one iteration defined by (1)—(3). It contains (from left to right) $q-1$ $IPSP$, a DP and $p-1$ $IPSP$ denoted by $L_1,\dots,L_{q-1}$, D and $R_{p-1},\dots,R_1$, respectively. The u-input and $\acute{u}$-output of R_j, are interchanged, $j=1,\dots,p-1$.

In the network only one of every two processors is active at any given time. If a processor is active then its neighbours are waiting. Each element $z_i(y_i)$ enter $L_1(R_1)$ as zero and it accumulates inner-product terms by moving to right (left), one term by processor. When z_i and y_i elements reach the proces-

sor D they have the value given by (1) and (2), respectively. The new value of x_i(*i.e.* $x_i^{(m+1)}$) computed by D following (3) goes leftward from L_j to L_{j-1}, $j=q-1,\dots,2$

The elements of $x^{(m)}$ enter R_{p-1} and they go from R_j to R_{j-1}, for $j=p-$
$-1,\dots,2$. The elements of b and of the main diagonal of A enter D.

The elements of vector $d_k=(a_{1,1+k}, a_{2,2+k},\dots,a_{n-k,n})$ are accepted by processor R_k, where $k=1,\dots,p-1$.

The processor L_{q-h} accepts the elements of $d'_h=(a_{h+1,1}, a_{h+2,2},\dots,a_{n,n-h})$, where $h=1,\dots,q-1$.

We shall analyse the manner that the activity of processors and the I/O operations are synchronized. Suppose that the z_i and y_i elements reach D at the $t_i\,CT$. It is obvious that $t_i=t_1+2(i-1)$, where the moment t_1 will be determined such that some initial conditions be satisfied.

The processor R_{p-k}, $k=1,\dots,p-1$, computes $y_i+a_{i,i+p-k}x_{i+p-k}$ at the $t_i-k\,CT$.

The element z_i is introduced as zero in L_1 at $tz(i)=t_i-q+1\,CT$.

At the tab $(1)=t_1\,CT$, the processor D begins to accept the pairs (b_i, a_{ii}), $i=1,\dots,n$, one pair at every two CT.

The element y_i is introduced as zero in R_1 at $ty(i)=t_i-p+1\,CT$.

The upper triangle of A is introduced as follows. The element $a_{i,i+p-k}$ enter R_{p-k} at the $t_i-k\,CT$, for $k=1,\dots,p-1$. Thus, the first element of d_j is received by R_j at the $td(j)=t_i-2i-p+j+2\,CT$, for $j=1,\dots,p-1$. Taking $j=1$ it results that the introducing of the upper triangle of A begins with a_{12} at $t_1-p+1\,CT$.

Observe that the processor L_{q-h}, $h=1,\dots,q-1$, computes $z_i+a_{i,i-h}\cdot x_{i-h}$ at the $t_i-h\,CT$. Thus, the processor L_{q-j} receives the first element of d'_j at the $td'(j)=t_i-2i+j+2\,CT$, where $j=1,\dots,q-1$. Taking $j=1$ it result that the introducing of the lower triangle of A begins with a_{21} at the $t_1+1\,CT$. Now, remark that at the $t_i-1\,CT$, the processor R_{p-1} receives the element x_{i+p-1}. If $tx(j)$ is the moment that x_j enter R_{p-1} then $tx(j)=t_{j-p+1}$. Because tx is an increasing function of j and the old value of x_1 is not used it results that the introducing of the old value of x_i begins with x_2 at the $t_{3-p}-1\,CT$.

Because $td'(1)>$tab(1) and either tab $(1)\geq ty(1)$ or tab $(1)\geq tz(1)$, the constant $c(p,q)=t_1$ will be determined so that min $(ty(1), tz(1), tx(2))=0$. Thus, we have $c(p,q)=\max\,(p-1, q-1, 2p-3)$.

After $c(p,q)+1\,CT$ the new value of x_1 can be extracted directly from D and the remaining x_1 exit from D at the rate of one every two CT. Therefore, the LSA computes $x^{(m+1)}$ in $c(p,q)+2n+1$ time units, as compared with $O(wn)$-time required for the usual sequential algorithm. If the nompenents of $x^{(m+1)}$ are extracted from L_1 then they appear after $c(p,q)+q+1$ with the same rate. As a result of the above-presented analyse we can state.

Theorem 1. *If A is a band matrix with width $w=p+q-1$ then the LSA containing w processors executes the iteration* (1)—(3) *with a response time equal to* $T_1(n)=c(p,q)+2n+1\,CT$.

It is obvious that for the presented LSA the host must broadcast the elements of A, b and $x^{(m)}$ at each new iteration. Without loss of generality, we shall suppose that the required number of M iterations is executed in M_s packets of s pipelining iterations, where $M_s=\min\,\{k/M\leq ks,\ k$ positive integer}. We want reduce the number of broadcastings to M_s. This task will be accomplished by the two-dimensional systolic array ($TDSA$) presented in Figure 3.

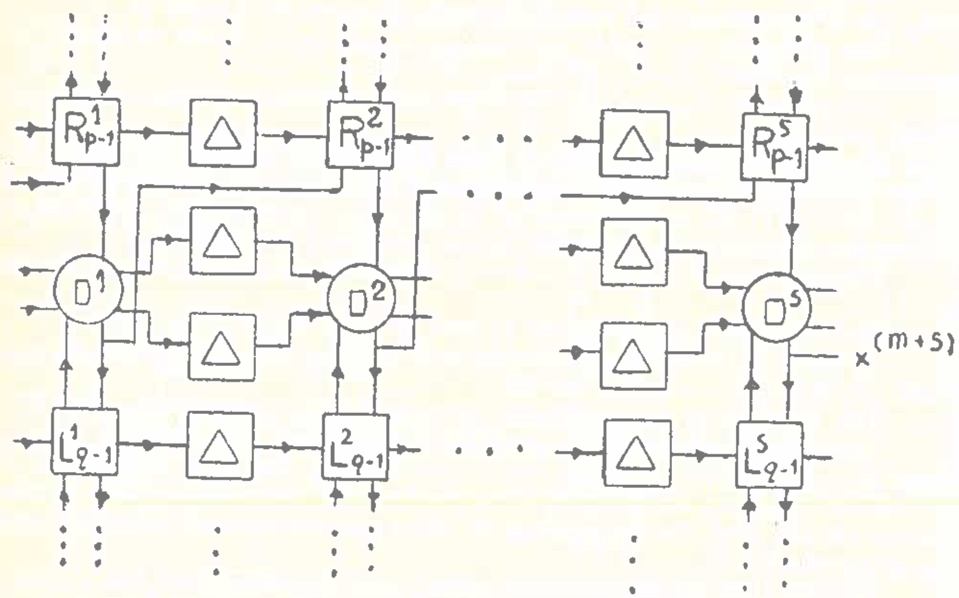

Figure 3. The T.D.S.A. that executes s pipelined iterations.

This system contains s connected LSA, $LSA^k=(L_1^k,\ldots,D^k,\ldots,R_1^k)$, $k=1,\ldots,s$ whose structure was already described. The data flow emerging from LSA^k enter LSA^{k+1}, for $k=1,\ldots,s-1$. All the flows introduced in LSA^{k+1} are synchronized as in the case of linear array, excepting the sequence $x^{(m+k)}$. Because $x_2^{(m+k)}$ appears in R_{p-1}^{k+1} later than the adequate moment, the remaining data flows must go through r delays. It suffices that the moments to introduce $x^{(m+k)}$ and the main diagonal of A are synchronized. Taking $k=1$ the disparity between $x_2^{(m)}$ and a_{11} is $3-2p\ CT$. The same disparity must be between $x_2^{(m+1)}$ and a_{11} in LSA^2. Because $x_2^{(m+1)}$ is given by LSA^1 at the $t_1+3\ CT$ and it is immediately transmitted to R_{p-1}^2, it results that the second disparity is $t_1+3-(t_1+r)$. Thus, we have $r=2p$. It results that LSA^k and LSA^{k+1} are interconnected using $p+q$ processors, for $k=1,\ldots,s-1$. Such a processor (denoted by a triangle) introduces a delay of $r\ CT$. It is clear that the resulting system remains systolic and it offers the approximate solution x^{m+1} after $c(p,q)+2n+1C$, while the remaining exit at the rate of one at every r units of time. Thus, the packet of s interations is executed after $c(p,q)+2n+1+(s-1)2p\ CT$. We can formulate

Theorem 2. *The TDSA with $w+(2w+1)(s-1)$ processors has the response time $T_2(s,n)=c(p,q)+2n+1+2p(s-1)$ and the period equal to $2p$, where A is a band matrix with width $w=p+q-1$.*

In the sequel we shall present a comparative analysis. Suppose that a number of s iterations is executed by (*i*) LSA and (*ii*) by $TDSA$. If $D_n(s)$ is the difference between the execution times obtained from (*i*) and (*ii*) then $D_n(s)=(s-1)\,(c(p,q)+2n+1-2p)$. It results that $D_n(s)>0$ for $s>1$ so that the pipelining of s iterations by $TDSA$ is more efficient than the use of LSA. Furthermore, $D_n(s)$ is an increasing function of n. The LSA has only w processors (of two types), while the $TDSA$ has $w+(2w+1)(s-1)$ processors of three types, but the processors that perform the delays are very simple. The advantage of both algorithms is that the hardware requirements are proportional to the width of the band and they do not depend on n. Now, if we solve $Ax=b$ using the LU-decomposition and the systolic systems have the same clock tick, then we need $T_{LU}(n)=7n+p+q+\min(p,q)\ CT$ and $pq+\max(p,q)$ processors. If m is the number of iterations (1)—(3) that have to be executed in order to obtain a desired error or the same accuracy as LU-decomposition, then the systolic variant of the Gauss-Seidel method will be more efficient if

$$m<sT_{LU}(n)/T_2(s,n) \tag{4}$$

for m a multiple by s, or

$$m<s(T_{LU}(n)-T_2(s',n))/T_2(s,n)+s' \tag{5}$$

for s' being the nonzero remainder of the division of m into s.

On the other hand, if the matrix inverter is used to solve $Ax=b$ (in the same assumption on the clock tick) we need $4n\ CT$ and n^2 processors. Because the matrix inverter does not exploit the kind of A, the additional operation $A^{-1}b$ takes $4n-3\ CT$ (using the matrix-vector multiplication system given in [4]). If these two stages are executed in a serial fashion then the total amount of time is $T_I(n)=8n-3\ CT$. Similar bounds for m may be obtained by putting $T_I(n)$ instead of $T_{LU}(n)$ in (4) and (5). A drawback of the matrix- inverter is that the hardware requirements depend on n^2.

3. A systolic variant of the acceleration method. In accordance with this method, the iteration is defined as

$$x_i^{(m+1)}=\omega(b_i-z_i-y_i)/a_{ii}+(1-\omega)x_i^{(m)}, \tag{6}$$

for $i=1,\ldots,n$ and $m\geq 1$, where $x^{(0)}$ is an arbitrary approximate solution, ω is an acceleration parameter and z_i and y_i are given by (1) and (2). A new linear systolic array can be obtained from LSA presented in the previous section by operating some minore changes.

The acceptance of the elements of $x^{(m)}$ starts with $x_1^{(m)}$ instead of $x_2^{(m)}$ with two CT before $x_2^{(m)}$. The DP must be able to keep the value of ω and compute the term $\omega\,(b_i-z_i-y_i)/a_{ii}$. This processor must have an additional input in order to accept the already calculated term $(1-\omega)x_i^m$. The discrepancy between the x_i-output of R_1 and the new input of DP is equal to $t_2-(tx(2)+p-1)=p$. Thus, we introduce two new processors, P_1 and P_2. Processor P_1 keeps the value $1-\omega$, accepts x_i from R_1, computes $(1-\omega)x_i$ and transmits this term to P_2. Processor P_2 transmits its input to DP after $p-1$ delays. If $p=1$ then P_2 is needless.

It is obvious that these changes do not affect the synchronization and the response time is $T_1(n)$. Some minor variations of this system may be obtained by introducing a new processor in order to avoid an overloading of DP.

A new two-dimensional systolic array can be obtained by applying the results presented in the previous section. The structure and the performances are preserved.

REFERENCES

1. Baudet, G. M. — *Design and complexity of VLSI algorithms*, Fundations of Computer Science, Part 1, Algorithms and Complexity, Mathematical Centre Amsterdam, 1983, pp. 49—74.
2. Foster, M. J., Kung, H. T. — *Design of special purpose VLSI chips: examples and opinions*, Techn. Rep. *CMU-CS-79-147*, Dept. of Computer Science, Carnegie Mellon University, Pittsburgh, 1979.
3. Kramer, M. R., Leeuwen, J. van — *Systolic computation and VLSI*, Fundations of Computer Science, Part, 1, Algorithms and Complexity, Mathematical Centre Amsterdam, 1983, pp. 75—103.
4. Leiserson, C. E. — *Area-Efficient VLSI Computation*, MIT Press, Massachusetts 1983.
5. Ullman, J. D. — *Computational aspects of VLSI*, Computer Science Press, Inc. 1984.

Received 28.III. 1985

Computer Center
Polytechnical Institute
6600 Iasi, Romania

Analele ştiinţifice ale Universităţii „Al. I. Cuza" din Iaşi
Tomul XXXI, s. I a, Matematică, 1985, f. 3.

ON A NEW KIND OF DEPENDENCIES IN RELATIONAL DATABASES

BY

C. MASALAGIU and H. LUCHIAN

§ 1. Introduction. We suppose the reader familiar with the usual approach for the theory of dependencies in relational databases([[4]], [[1]], [[7]], [[10]]).

Let $\mathcal{U}$ be a denumerable set of *attributes* and $M=\{., c\}$ the set of marks. Let $\mathcal{U}_M=\mathcal{U}\times M$.

In the sequel, we shall use the following notations: $[m]$ will denote the set $\{1, 2,\ldots,m\}$, $m\in N$; for $a=(u, m)\in\mathcal{U}_M$, we shall consider supp $(a)=u$ and for $\{a_1,\ldots,a_n\}\subseteq\mathcal{U}_M$, supp $(\{a_1,\ldots,a_n\})=\{\text{supp }(a_1),\ldots,\text{supp }(a_n)\}$; for $\{A_1,\ldots,A_m\}\subseteq\mathcal{U}$ we shall use the shorthand $A_1 \ldots A_m$.

A *relation scheme* over $\mathcal{U}$ is any finite subset of $\mathcal{U}$. From now on, we shall reffer to a fixed relation scheme $R=\{A_1,\ldots, A_n\}$, $n\geqslant 2$; $R\times M$ will be denoted by R_M.

For each $A\in\mathcal{U}$, let the set D_A be the *domain* of A; card $D_A\leqslant\aleph_0$. For each $a\in\mathcal{U}$, let $\alpha_A\subseteq D_A\times D_A$ be a reflexive, nontrivial relation ($\alpha_A\neq D_A\times D_A$).

Let $\text{Tup}_R\mathcal{U}=\{u/u : [n]\to\bigcup_{i=1}^{n} D_{A_i}, u(\mathrm{i})\in D_{A_i}\}$ be the set of tuples over R (we shall alternatively use $u(i)$ or $u(A_i)$, D_i or D_{A_i}, α_i or α_{A_i}, a.s.o.).

Let Q be any set $[n]$ or N itself. A *Q-instance* for R is an injective function $I_R : Q\to\text{Tup}_R\mathcal{U}$. The set of all Q-instances for R will be denoted by $\mathcal{I}_{R, Q}$.

An *instance* for R is a Q-instance, for some Q. If I is a Q-instance for R and $Q_1\subseteq Q$, any Q_1-instance I_1: $Q_1\to\text{Tup}_R\mathcal{U}$ such that $I/Q_1=I_1$, will be called *subinstance* of I (Q_1 will be either any $[n]$, or N itself). The set of all instances for R is denoted by $\mathcal{I}_R$.

Remark. The instances in the definition above appear to be ordered sets of tuples; we note that this „ordering" depends on what ordering (if any!) is considered on N. A forth coming paper will examine closely the necessity of changing some classical definitions. For the moment let us only note that there are cases where ordered instances must be explicitely taken into account.

As usual, only a subset of $\mathcal{I}_R$ will be considered, which we shall call *the set of admissible instances* ($\mathcal{A}_R$). The admissible instances will obey certain specific requirements for $Im(I_R)$, $I_R\in\mathcal{I}_R$. For example: $\mathcal{A}_R^1=\bigcup_{m\geqslant 2}\mathcal{I}_{R, [m]}$, $A_R^2=\bigcup_{m\leqslant n}\mathcal{I}_{R, [m]}$, $\mathcal{A}_R^3=\mathcal{I}_{R, N}$, $\mathcal{A}_R=\mathcal{A}_R^1\cup\mathcal{A}_R^3$, a.s.o.