Discrete Mathematics and Theoretical Computer Science **5**, 2002, 109–120

Synthesis Of Space-Time Optimal Systolic Algorithms For The Cholesky Factorization†

Clémentin Tayou Djamegni

Laboratoire d'Informatique, Faculté des Sciences, Université de Dschang B.P. 069 Dschang Cameroun
E-mail: `dtayou@cril.univ-artois.fr`

received Apr 23, 2001, *revised Feb 19, 2002*, *accepted Mar 25, 2002.*

In this paper we study the synthesis of space-time optimal systolic arrays for the Cholesky Factorization (CF). First, we discuss previous allocation methods and their application to CF. Second, stemming from a new allocation method we derive a space-time optimal array, with nearest neighbor connections, that requires $3N+\Theta(1)$ time steps and $N^2/8+\Theta(N)$ processors, where N is the size of the problem. The number of processors required by this new design improves the best previously known bound, $N^2/6+\Theta(N)$, induced by previous allocation methods. This is the first contribution of the paper. The second contribution stemms from the fact that the paper also introduces a new allocation method that suggests to first perform index transformations on the initial dependence graph of a given system of uniform recurrent equations before applying the weakest allocation method, the projection method.

Keywords: Parallel processing, projection methods, timing function, allocation function, space-time complexity, re-indexation

1 Introduction

The Cholesky Factorization (CF) is an important problem in computer science. Algorithms such as CF are kernels of many numeric or signal processing programs. Because of the large number of arithmetic operations, $\Theta(N^3)$ where N is the size of the matrix, required by the CF a number of research works have been devoted to its parallelization [5, 11, 17]. In [17] Liu identifies three potential levels of *granularity* in a parallel implementation of the CF:

1. *fine-grain*, in which each task consists of only on one or two floating point operations. Fine-grain parallelism is available either for dense or sparse matrices.

2. *medium-grain*, in which each task is a single column operation. Medium-grain parallelism is also available either for dence or sparse matrices.

†This work has been supported in part by the *Université d'Artois*, the *Nord/Pas-de-Calais Région* under the TACT-TIC project, the European Community FEDER Program, the French Agency *Aire Développement* through the project *Calcul Parallèle* and by the *Microprocessors and Informatics* Program of the United Nations University.

3. *large-grain*, in which each task is the computation of an entire group of columns. Large-grain parallelism, at the level of subtrees of the elimination tree [17], is available only in the sparse case.

Hereafter we focus on fine-grain parallelism. This parallelism can be exploited effectively by massively parallel computers such as SIMD computers and systolic arrays.

Systolic arrays as introduced by Kung and Leiserson [13] form an attractive class of special-purpose parallel architectures suitable for an implementation in VLSI. A systolic array consists of a large number of elementary processors, also called processing element (PE), which are regularly connected in a nearest neighbor fashion. Each PE is equipped with a limited storage and operates on a small part of the problem to solve. As VLSI enables inexpensive special-purposes chips, systolic arrays are typically used as back-end, special-purpose devices to meet high performance requirements. A number of such arrays are currently used to accelerate computations arising in bioinformatics, biology and chemistry (http://www.irisa.fr/cosi/SAMBA, http://www.timelogic.com, http://www.paracel.com, http://www.cse.ucsc.edu/research/kestrel) among others.

Most of the early systolic arrays were designed in an ad hoc case-by-case manner. But, this case-by-case approach requires an apostoriori verification to guarantee the correctness of the resulting design. In recent years there has been a great deal of effort on developping unifying theories for automatically synthesizing such arrays. It is well known that the standard methodology [18, 20, 21, 22, 23, 25] for the systematic synthesis of systolic array proceeds in four points:

1. The starting point is a solution of the problem to solve in term of a system of recurrent equations (SREs).

2. The second point deals with the uniformization of the SREs. This point leads to a system of uniform recurrent equations (SUREs) associated to a dependence graph $G = (D,U)$ in which each node corresponds to a task and each link corresponds to a dependency between two elementary computations, also called task.

3. The third point defines a timing function (or schedule) $t : D \rightarrow N$ which gives the computation date $t(v)$ of each task v of D, assuming that all tasks are of unit delay. A schedule is optimal if the corresponding execution time (or time steps count) is the length of the longest path of the dependence graph G.

4. The last point defines an allocation function which assigns the tasks to the processors of a systolic array so as to avoid computation conflict, i.e no two tasks with the same execution date should be executed by the same PE. An allocation function is optimal if the PEs count of the resulting array is the *potential parallelism* , i.e the maximum number of tasks that are scheduled to be executed at the same date.

The standard methodology poses a number of interesting optimization problems.

1. The problem of how to find a linear schedule that minimizes the execution time over all possible linear schedules associated to a given SUREs has been investigated by several authors. Approaches, based on integer programming, to a general solution are proposed in [26, 32]. Note that the resulting linear schedule may not correspond to an optimal schedule.

2. The problem of transforming a SREs into a SUREs that leads to a timing function that requires the minimum possible execution time has attracted special interest. In other words, the question is to minimize the execution time independently of the uniformization of the initial SREs. This question remains open although there are now well defined methods [21, 22, 23, 35] to systematically transform most *Affine Recurrent Equations* (AREs) into a SUREs. This open question is investigated in [27], by Djamegni et al., for the *Algebraic Path Problem* (APP). In their paper the authors introduce a new uniformization technique that transforms the initial SREs (associated to the APP) into a new SUREs. The new SUREs leads to a piecewise affine schedule of $4n+\Theta(1)$ steps, where n is the size of the APP. This is a significant improvement over the number of steps, $5n+\Theta(1)$, of schedules induced by the earlier uniformization technique [1].

3. The problem of finding an optimal timing function for a given SUREs so as to minimize the paralelism rate has not yet received a full answer. In other words, the question is to minimize the potential parallelism independently of the timing function. Integer progamming techniques can be used to find the best affine schedule possible. However, the time spent to find such an affine schedule cannot be neglected [7]. This open question is investigated in [28], by Djamegni et al., for the Triangular Matrix Inversion (TMI). These authors have proposed an optimal piecewise affine timing function with a potential parallelism of the order of $n^2/8+\Theta(n)$, where n is the size of the triangular matrix. This is a significant improvement over the minimal potential parallelism $n^2/6+\Theta(n)$ provided by optimal affine timing functions.

4. The probem of defining an allocation function that minimizes the PEs count for a given schedule remains open. Research on this optimization criterion have been presented in [1, 2, 3, 5, 6, 25, 27, 28, 29, 30, 31, 33, 34] among others.

In this paper, we are interested in the design of a space-time optimal systolic array for the CF. The main difficulty of this problem is to meet the last (fourth) optimization criterion. In the following we briefly sommarize previous works related to this design constraint.

1. *Projection Method* [18, 20, 21]. This method corresponds to linear allocations. Such allocations are realized by projecting the dependence graph G along a direction $\vec{p}$. All the tasks belonging to a same line of direction $\vec{p}$ are assigned to the same PE. The main drawback of this allocation technique is the low PE utilization occuring in the resulting array [5]. Wong and Delosme [34], Ganapathy and Wah [9] use integer programming to get the best linear allocation possible. However, such a linear allocation may not correspond to an optimal allocation. For the CF the minimal PEs count obtained by projecting the dependence graph is $N^2/2+\Theta(N)$.

2. *Grouping (or Clustering) Method* [4, 6, 7]. The starting point is a systolic array obtained from the projection method in which all PE works one step over x, with $x \geq 2$. Then, groups of x neighboring PEs having disjoint working time steps are grouped into a single one. As a consequence, the PEs count of the initial array is reduced by a factor of x. This allocation technique do not guarantee space-optimality. For the CF this method permits to reduce the PEs count from $N^2/2+\Theta(N)$ to $N^2/4+\Theta(N)$.

3. *Instruction Shifts Method* [5]. As in the grouping method, the starting point is an array obtained from the projection method. Then, the initial array is partionned into a number of PEs segments

parallel to a direction called *partition direction*. The tasks of each segment are reallocated so as to minimize the PEs count on each segment. This approach do not guarantee space-optimality. For the CF this technique reduces the PEs count from $N^2/2+\Theta(N)$ to $N^2/6+\Theta(N)$ [5]. Although this represents a significant improvement, this design is not space optimal as the potential parallelism of the CF is $N^2/8+\Theta(N)$ [5].

4. *Piling Method* [1, 6]. First, from a given dependence graph and affine schedule, a set M of tasks is found such that all tasks in the set are scheduled to be executed at the same time and the set size $\mid M \mid$ is maximal. Second, an allocation method is applied to assign the tasks of the dependence graph to PEs. Any PE which has not been assigned to execute a task of M is piled to a PE which executes a task of M and has disjoint working time steps. However, piling PEs results in long range communications such as spiral links and increases irregularity for the resulting arrays. In this paper, we seek to avoid this.

5. *Partition Method* [25, 31]. The starting point is to partition the dependence graph G into a number of sub-graphs of lower dimension. Then the tasks of each sub-graph are allocated to PEs so as to minimize the PEs count on each sub-graph. This allocation heuristic leads to an array of $n^2/8+\theta(n)$ PEs for the *Dynamic Programming* (DP), where n is the size of the problem [15]. In [29] Djamegni et al. reduces the PEs count of the DP to $n^2/10+\theta(n)$. This better solution is obtained by merging nodes of the dependence graph associated with de DP before applying the partition method. All these solutions are not space-optimal as they are based on the earliest optimal timing function whose the potential parallelism is $n^2/14+\Theta(n)$. The partition method is also used in [27] by Djamegni et al. to derive a space-optimal array for the APP and in [2, 3] by Bermond et al. to derive various arrays for the *Gaussian Elimination* (GE). However these GE solutions are not space-optimal. A Space-time optimal array for the GE is proposed in [1] by Benaini and Robert. For CF the partition method leads to an array of $N^2/6+\Theta(N)$ PEs.

The weakest allocation technique, in term of the resulting PEs count, seems to be the projection method. The most interesting seem to be the instruction shifts method and the partition method. Combination of different allocation techniques is possible. For instance, in [28] Djamegni et al. show how one can combine projection, piling and partition methods to design a space-optimal array for the TMI when the schedule corresponds to a piecewise affine timing function. However, their design do not correspond to a systolic array because of long-range communications occuring in the array.

We believe that it is important to investigate the problem of designing space-optimal arrays based on domain transformations, given that: (i) there are tools that propose to (semi) automatically generate application specific VLSI processor arrays, (ii) such arrays are becoming more and more powerful (iii) systolic array synthesis methods find applications in parallelizing compilers. For this last point, the PAF (*Paralléliseur Automatique de Fortran*) project [8] uses a generalization of systolic schedule and allocation techniques for generating parallel code. The LooPo project [10] at the University of Passau explores parallelization with systolic synthesis methods, as does the OPERA project [19] at the University of Strasbourg. Systolic schedule and allocation techniques are also used in [11] to compute CF on distributed memory parallel computers.

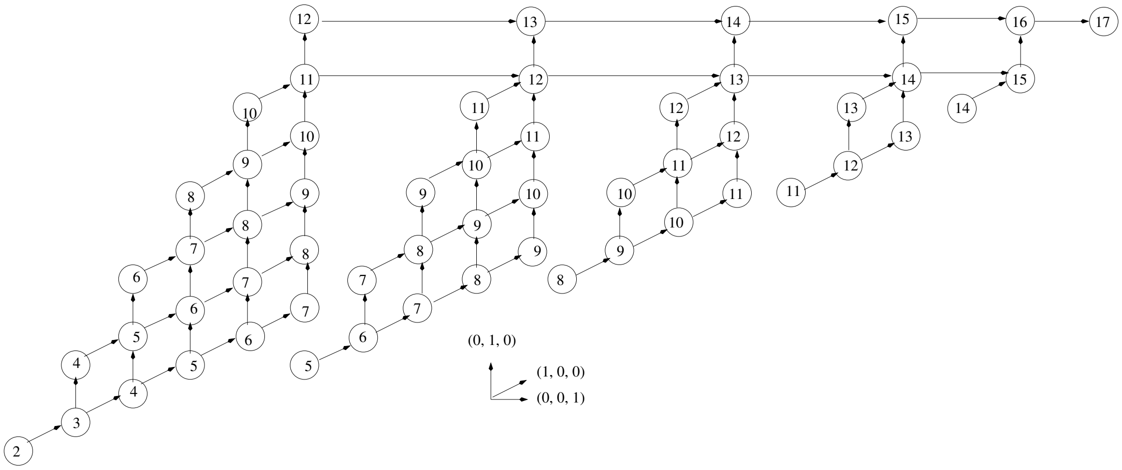

Fig. 1: The dependence graph of the CF for $N = 6$

In this paper, we derive a space-time optimal systolic array for the CF that requires $3N + \Theta(1)$ time steps and $N^2/8 + \Theta(N)$ PEs. This constitutes the first contribution of the paper. The second contribution stemms from the fact that this new array is obtained from a new allocation strategy that suggests to re-index the initial dependence graph of the CF before applying the weakest allocation method, the projection method. As this new allocation strategy is based on re-indexing transformations, it could be integrated in parallelizing compilers and tools.

Throughout this paper we will use notation $[z \rightarrow z\prime]$ which expresses a causal dependency between the index points z and $z\prime$, i.e results of calculations associated to point z are needed by calculations assigned to point $z\prime$. We will also use the following definition: a plane is said to be parallel to direction $< \vec{a},\ \vec{b} >$ if it is parallel to vectors $\vec{a}$ and $\vec{b}$.

The rest of the paper is organized as follows. Section 2 discusses the application of previous allocations methods to CF. Section 3 presents our contribution. This section applies a new allocation strategy to the CF, and this results in a space-time optimal array that improves previous solutions. Concluding remarks are stated in the last section.

2 Deriving Systolic Arrays From Previous Allocation Techniques

The CF is defined as follows: Given a $N \times N$ symetric positive definite matrix A, the CF calculates a lower triangular matrix L such that $A = LL^t$. It is defined by the following well known affine recurrence equations:

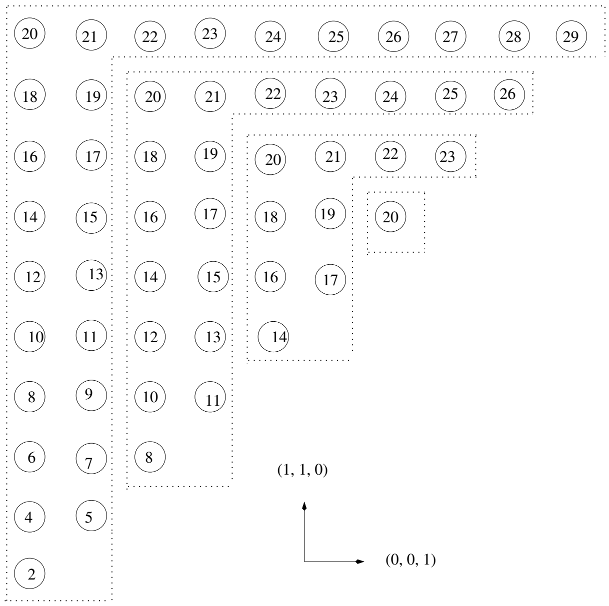

Fig. 2: Illustration of the partition method on G_1 for $N = 10$

For $(i,j,k) \in D = \{1 \le j \le i \le N \ \wedge \ 0 \le k \le j\}$

$$L(i,j,k) = \begin{cases} A_{j,i} & \text{if } 1 \le j \le i \le N \wedge k = 0 \\ L(i,j,k-1)/L(j,j,j-1) & \text{if } 1 \le i \le N \wedge 1 \le j \le i-1 \wedge k = j \\ L(i,j,k-1)^{1/2} & \text{if } 1 \le i \le N \wedge j = i \wedge k = i \\ L(i,j,k-1) - L(i,k,k) & \text{if } 1 \le j \le i \le N \wedge 1 \le k \le j-1 \\ \qquad\qquad \times L(j,k,k) & \end{cases} \tag{1}$$

Following the standard methodology for the systematic synthesis of systolic architectures [18, 20, 21, 22, 23, 25] we first derive a uniform version of (1). Regarding the dependencies of (1) a uniform version S can be obtained with $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$ as the dependence vectors [21], and this without changing the domain D of equations (1). An optimal timing function corresponding to such a uniformization is $t(i,j,k) = i + j + k$. The dependence graph G and the timing function are illustrated in figure 1. In this figure, we have not draw all the dependence vectors of direction $(0,0,1)$ for sake of clarity.

By projecting the domain D of equations S along vector $\vec{p1} = (1,1,0)$, we obtain a triangular orthoganally connected array of $N(N+1)/2$ processors which solves the CF in optimal time $3N + \Theta(1)$. In the resulting array, all PE is active only once over two time steps. Using the grouping method, the array can be compressed by a factor of 2, thereby reducing the PEs count to $N^2/4 + \Theta(N)$. Note that if we project D along vector $\vec{p2} = (1,0,0)$ we also obtain a triangular orthoganally connected array of $N(N+1)/2$ processors. But this last array can not be compressed by the grouping method as all PE is active at any time step between its first and last calculations. On the other hand, the space complexity of this array has been reduced to $N^2/6+\Theta(N)$ [5] by applying the instruction shifts method with the partition direction $\vec{a}=(0,1,1)$. This is a significant improvement over the space complexity of the initial array. However this array is not space-optimal as the potential parallelism of the problem is $N^2/8+\Theta(N)$ [5]. Now let apply the partition method [25, 31]. To this end, we first consider the intersections of the dependence graph G with a number of planes of direction $<(1,1,0),\ (0,0,1)>$. This partitions G into N sub-graphs, $G_h=\{(x,y,z)\in G \mid x-y=h-1\},\ h=1,2,\ \dots\ N$. Then the tasks of each sub-graph are seperately allocated so as to minimize the PEs count for each sub-graph as in [16], where it is proposed a space-time optimal systolic array of $n^2/6+\Theta(n)$ PEs for TMI (n is the size of the triangular matrix). Figure 2 illustrated how the tasks of a sub-graph are allocated to PEs. The idea behind this allocation is to recursively assign at each step two columns of direction $(1,1,0)$ and one line of direction $(0,0,1)$ to a new PE. Sub-graph G_h induces $\lceil (N-h+1)/3\rceil$ PEs. Thus the overall PEs count is $N^2/6+\Theta(N)$. Note that if the partition method is applied by partitioning G following direction $<\vec{a},\ \vec{b}>$ where $\vec{a}$ and $\vec{b}$ belong to $\{(1,0,0),\ (0,1,0),(0,0,1)\}$ ($\vec{a}\neq\vec{b}$), the resulting array will required $N^2/4+\Theta(N)$ PEs [25, 31].

3 Our Contribution: A Space-Time Optimal Design

In this section we derive a new systolic array of $N^2/8+\Theta(N)$ PEs which solves the CF problem in $3N+\Theta(1)$ steps. This is space-time optimal as the potential parallelism is $N^2/8+\Theta(N)$ [5]. This better solution is obtained by performing a pre-processing by re-indexation which transforms the domain D of equations S into a new one which is more suitable for the application of projection methods.

Note that we can assume without loss of generality that the input (resp. output) points of equations S are of the form $(i,j,0)$ (resp. (i,j,j)), with $1\leq j\leq i\leq N$. In the following we present a space-time optimal systolic array with nearest neighbor connections for the CF problem. Note that we make a distinction between nearest-neighbor arrays and local arrays where the interconnections may "jump" over one or more processors (the so-called bounded broadcast facility [24, 35]). This often allows a constant factor of improvement, and the method can be applied to the array that we present. The derivation of this new design can be divided into three phases.

Phase 1 We first apply to system S an unimodular transformation q that transforms the initial affine timing function $t(i,j,k)=i+j+k$ into a new one $t\prime(i,j,k)=i$. To do so we set $q(i,j,k)=(i+j+k,j,k)$. The re-indexing q leaves inchanged the initial dependence vector $(1,0,0)$ while $(0,1,0)$ and $(0,0,1)$ become $(1,1,0)$ and $(0,1,1)$ respectively. It also maintains input points (resp. output points) on their initial plane, i.e plane of cartesian equation $k=0$ (resp. $-j+k=0$). Denote $D^{(0)}=\{(i,j,k)\in Z^3 \mid -i+2j+k\leq 0,\ i-j-k-N\leq 0,\ -k\leq 0,\ -j+k\leq 0\}$. The domain of the new system $q(S)$ is a subset of $D^{(0)}$. In the following phase we consider $D^{(0)}$ to simplify the presentation.

Phase 2 We now apply to $D^{(0)}$ a re-indexation that locates the points of $\{(i,j,k)\in D^{(0)} \mid i-j-k-N=0\}$ on the plane of cartesian equation $k=0$. To do so, we use the re-indexing function q_0 defined by:

$$q_0(i,j,k)=\begin{cases}(i,j,-i+j+k+N) & \text{if } (i,j,k)\in D_1^{(0)}\\ (i,j,k) & \text{if } (i,j,k)\in D_2^{(0)}\end{cases}$$

where

$$D_1^{(0)}=\{(i,j,k)\in D^{(0)} \mid -i+j+N<0\}$$

$$D_2^{(0)} = \{(i,j,k) \in D^{(0)} \mid -i+j+N \geq 0\}$$

Denote $Q_1^{(0)} = q_0(D_1^{(0)})$, $Q_2^{(0)} = q_0(D_2^{(0)})$ and $D^{(1)} = q_0(D^{(0)}) = Q_1^{(0)} \cup Q_2^{(0)}$. We have $Q_1^{(0)} = \{(i,j,k) \in Z^3 \mid -i+j+N < 0,\ j+k-N \leq 0,\ -k \leq 0,\ i-2j+k-N \leq 0\}$ and $Q_2^{(0)} = \{(i,j,k) \in Z^3 \mid -i+j+N \geq 0,\ -i+2j+k \leq 0,\ -k \leq 0,\ -j+k \leq 0\}$. Simple linear algebraic shows that $(i,j,k) \in Q_1^{(0)}$ implies $-j+k \leq 0 \ \wedge\ -i+2j+k \leq 0$ and that $(i,j,k) \in Q_2^{(0)}$ implies $j+k-N \leq 0 \ \wedge\ i-2j+k-N \leq 0$. Thus $D^{(1)} = \{(i,j,k) \in Z^3 \mid j+k-N \leq 0,\ -k \leq 0,\ i-2j+k-N \leq 0,\ -i+2j+k \leq 0,\ -j+k \leq 0\}$.

The re-indexing q_0 acts differently on the dependencies of the two regions $D_1^{(0)}$ and $D_2^{(0)}$ of $D^{(0)}$. It transforms dependencies $[z-(1,0,1) \longrightarrow z]$, $[z-(1,1,0) \longrightarrow z]$ and $[z-(1,0,0) \longrightarrow z]$ into $[q_0(z)-(1,0,0) \longrightarrow q_0(z)]$, $[q_0(z)-(1,1,0) \longrightarrow q_0(z)]$ and $[q_0(z)-(1,0,-1) \longrightarrow q_0(z)]$ respectively if $z \in D_1^{(0)}$, and into $[q_0(z)-(1,0,1) \longrightarrow q_0(z)]$, $[q_0(z)-(1,1,0) \longrightarrow q_0(z)]$ and $[q_0(z)-(1,0,0) \longrightarrow q_0(z)]$ respectively if $z \in D_2^{(0)}$. The resulting dependence vectors are: $(1,0,0)$, $(1,1,0)$ and $(1,0,-1)$ in region $Q_1^{(0)}$, and $(1,0,1)$, $(1,1,0)$ and $(1,0,0)$ in region $Q_2^{(0)}$.

Note that the re-indexing q_0 leaves inchanged the timing function $t\prime(i,j,k) = i$. Thus vector $\vec{p} = (1,0,0)$ is a valid projection direction. It leads to a triangular orthogonally connected array of $N^2/4 + \Theta(N)$ processors which is two times smaller than the size complexity of the array obtained by projecting the initial domain D along the same direction. On the other hand, the re-indexing q_0 maintains the input points (resp. output points belonging to $D_2^{(0)}$) on the plane of cartesian equation $k = 0$ (resp. $-j+k = 0$) and locates the output points belonging to $D_1^{(0)}$ on the plane of equation $i-2j+k-N = 0$.

Phase 3 This array can be further optimized. For this purpose we re-index the points of $D^{(1)}$ so as to locate the points of $\{(i,j,k) \in D^{(1)} \mid j-k = 0 \ \vee\ i-2j+k-N = 0 \ \vee\ i-2j+k-N = 1\}$ on the plane of cartesian equation $j = 0$. To do so, we consider the re-indexing function q_1 defined by:

$$q_1(i,j,k) = \begin{cases} (i, -\frac{1}{2}i + j - \frac{1}{2}k + \frac{N}{2}, k) & \text{if } (i,j,k) \in D_{1,0}^{(1)} \\ (i, -\frac{1}{2}i + j - \frac{1}{2}k + \frac{N+1}{2}, k) & \text{if } (i,j,k) \in D_{1,1}^{(1)} \\ (i, j-k, k) & \text{if } (i,j,k) \in D_2^{(1)} \end{cases}$$

where

$$D_{1,0}^{(1)} = \{(i,j,k) \in D^{(1)} \mid -i+j+N < 0 \wedge (i+k-N)(\bmod 2) = 0\}$$

$$D_{1,1}^{(1)} = \{(i,j,k) \in D^{(1)} \mid -i+j+N < 0 \wedge (i+k-N)(\bmod 2) = 1\}$$

$$D_2^{(1)} = \{(i,j,k) \in D^{(1)} \mid -i+j+N \geq 0\}$$

Denote $D_1^{(1)} = D_{1,0}^{(1)} \cup D_{1,1}^{(1)}$, $D^{(2)} = q_1(D^{(1)})$, $Q_1^{(1)} = q_1(D_1^{(1)})$, $Q_{1,0}^{(1)} = q_1(D_{1,0}^{(1)})$, $Q_{1,1}^{(1)} = q_1(D_{1,1}^{(1)})$, $Q_2^{(1)} = q_1(D_2^{(1)})$, $\alpha_0 = 0$ and $\alpha_1 = \frac{1}{2}$. We have $Q_{1,h}^{(1)} = \{(i,j,k) \in Z^3 \mid i+2j+3k-3N-2\alpha_h \leq 0,\ -j \leq 0,\ -k \leq$

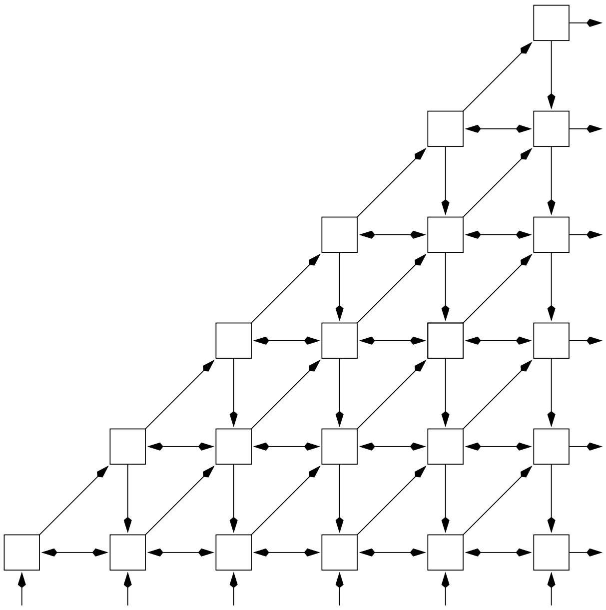

Fig. 3: Array obtained for $n = 12$

$0,\ 2j+2k-N-2\alpha_h \le 0,\ i-k+N<0\}$ where $h \in \{0,\ 1\}$, and $Q_2^{(1)} = \{(i,j,k) \in Z^3 \mid j+2k-N \le 0,\ -k \le 0,\ -i+2j+3k \le 0,\ -j \le 0,\ -i+k+N \ge 0\}$. Clearly, $Q_1^{(1)} = E \cup F$ where $E = \{(i,j,k) \in Z^3 \mid i+2j+3k-3N \le 0,\ -j \le 0,\ -k \le 0,\ 2j+2k-N \le 0,\ -i+k+N < 0\}$ and $F = \{(i,j,k) \in Q_{1,0}^{(1)} \mid i+2j+3k-3N = 1$ or $2j+2k-N = 1\}$. Simple linear algebraic shows that $(i,j,k) \in E$ implies $i+2j+3k-3N \le 0 \wedge j+2k-N \le 0$ and that $(i,j,k) \in Q_2^{(1)}$ implies $i+2j+3k-3N \le 0 \wedge j+2k-N \le 0$. Thus $D^{(2)} = F \cup \{(i,j,k) \in Z^3 \mid -k \le 0,\ -j \le 0,\ i+2j+3k-3N \le 0,\ 2j+2k-N \le 0,\ -i+2j+3k \le 0$ and $j+2k-N \le 0\}$.

Let's now determine the new dependencies induced by the re-indexing q_1. For this purpose, we first partition $D^{(1)}$ into three regions R_1, R_2 and R_3. From $D^{(1)} = Q_1^{(0)} \cup Q_2^{(0)}$ and $D^{(1)} = D_1^{(1)} \cup D_2^{(1)}$ we get $D^{(1)} = (Q_1^{(0)} \cup Q_2^{(0)}) \cap (D_1^{(1)} \cup D_2^{(1)}) = (Q_1^{(0)} \cap D_1^{(1)}) \cup (Q_2^{(0)} \cap D_1^{(1)}) \cup (Q_2^{(0)} \cap D_2^{(1)})$ as $Q_1^{(0)} \cap D_2^{(1)}$ is empty. We set $R_1 = Q_1^{(0)} \cap D_1^{(1)}$, $R_2 = Q_2^{(0)} \cap D_1^{(1)}$ and $R_3 = Q_2^{(0)} \cap D_2^{(1)}$. Note that in region R_i, the dependencies are: $[z-(1,1,0) \longrightarrow z]$, $[z-(1,0,0) \longrightarrow z]$, $[z-(1,0,-1) \longrightarrow z]$ if $i=1$ and $[z-(1,0,1) \longrightarrow z]$ if $i = 2$ or $i = 3$, with $z \in R_i$. We now explain how these dependencies are transformed per region by the re-indexing q_1.

Case of Region R_1. The dependency $[z-(1,1,0) \longrightarrow z]$ become $[q_1(z)-(1,0,0) \longrightarrow q_1(z)]$ if $z \in D_{1,0}^{(1)}$ and $[q_1(z)-(1,1,0) \longrightarrow q_1(z)]$ if $z \in D_{1,1}^{(1)}$. The dependency $[z-(1,0,0) \longrightarrow z]$ become $[q_1(z)-(1,-1,0) \longrightarrow q_1(z)]$ if $z \in D_{1,0}^{(1)}$ and $[q_1(z)-(1,0,0) \longrightarrow q_1(z)]$ if $z \in D_{1,1}^{(1)}$. The dependency $[z-(1,0,-1) \longrightarrow z]$ become

$[q_1(z) - (1,0,-1) \longrightarrow q_1(z)]$ if $z \in R_1$.

Case of Region R_2. The dependencies $[z - (1,1,0) \longrightarrow z]$ and $[z - (1,0,0) \longrightarrow z]$ are transformed as in region R_1, while the dependency $[z - (1,0,1) \longrightarrow z]$ is transformed into $[q_1(z) - (1,-1,1) \longrightarrow q_1(z)]$.

Case of Region R_3. The dependencies $[z - (1,1,0) \longrightarrow z]$, $[z - (1,0,0) \longrightarrow z]$ and $[z - (1,0,1) \longrightarrow z]$ become $[q_1(z) - (1,1,0) \longrightarrow q_1(z)]$, $[q_1(z) - (1,0,0) \longrightarrow q_1(z)]$ and $[q_1(z) - (1,-1,1) \longrightarrow q_1(z)]$ respectively.

The resulting dependence vectors on image $q_1(R_i)$ of region R_i by the re-indexing q_1 are: $(1,1,0)$, $(1,0,0)$, $(1,-1,0)$ if $i = 1$ or $i = 2$, $(1,-1,1)$ if $i = 2$ or $i = 3$, and $(1,0,-1)$ if $i = 1$.

We are now ready to derive a space-time optimal solution using projection method. For this purpose we project domain $D^{(2)}$ along direction $\vec{p} = (1,0,0)$. The resulting array is illustrated in figure 3. It corresponds to a triangular hexagonally connected array of $N^2/8 + \Theta(N)$ processors which is two times smaller than the size complexity of the array obtained in phase 2. Moreover this new design is space-optimal as the potential parallelism of the problem is [5] $pp = N^2/8 + \Theta(N)$. On the other hand, the re-indexing q_1 maintains the input points on the plane of cartesian equation $k = 0$ and moves the output points from planes $-j + k = 0$ and $i - 2j + k - N = 0$ to plane $j = 0$. This implies that input and output points are allocated to processors located at the border of the array. A property which permit to avoid eventual additional steps for data loading and result unloading. Thus this array solves CF problem in optimal time $3N + \Theta(1)$. Therefore this new design is space-time optimal.

4 Conclusion

In this paper we have studied various allocation methods and their application to CF. In particular, we have designed a systolic array with regular nearest-neighbor connections for the CF problem that is both time-optimal and space-optimal, thereby establishing the "systolic complexity" of the CF problem. We believe that the design of a space-time optimal array represents an interesting and important contribution to the knowledge of the systolic model. This space-time optimal solution is obtained from a new allocation strategy that suggests to first perform index transformations on a dependence graph before applying the weakest allocation method, the projection method. However, the main difficulty of this allocation stragegy stemms from the definition of suitable index transformations. On the other hand, we believe that tools that permit to (semi)automatically generate application specific VLSI processors arrays, research on the parallelization of algorithms for distributed-memory massively parallel computers [11] and research on parallelizing compilers [8, 10, 19] could benefit from systolic allocation strategies based on such index transformations.

Acknowledgements
The author would like to thank DMTCS for multiple useful remarks and suggestions.

References

[1] A. Benaini and Y. Robert, Space-minimal systolic arrays for Gaussian elimination and the algebraic path problem. *Parallel Computing*, 15 (1990) 211-225.

[2] J.C. Bermond, C. Peyrat, I. Sakho and M. Tchuente, Parallelisation of the Gaussian elimination on systolic arrays. *Journal of Parallel and Distributed Computing*, 33 (1996) 69-75.

[3] J.C. Bermond, C. Peyrat, I. Sakho and M. Tchuente, Parallelisation of the Gauss elimination on systolic arrays. *Internal Report, LRI*, 430 (1988).

[4] J. Bu and E.F. Deprettere, Processor clustering for the design of optimal fixed-size systolic arrays. In Proc. *Sixth Int'l Parallel Processing Symp.*, (Mar. 1992) 275-282.

[5] P. Clauss. Optimal mapping of systolic algorithms by regular instruction shifts. *IEEE Int. Conf. on Application Specific Array Processors*, (1990) 224-235.

[6] P. Clauss, C. Mongenet and G. R. Perrin. Calculus of space-optimal mappings of systolic algorithms on processor arrays. *IEEE Int. Conf. on Application Specific Array Processors* , (1990) 591-602.

[7] A. Darte, T. Risset and Y. Robert, Synthesizing systolic arrays: Some recent developments. *Int. Conf. on Application Specific Array Processors, IEEE Computer Soc. Press*, (1991) 372-386.

[8] P. Fautrier, Dataflow analysis of array and scalar references. *Internationnal Journal Of Parallel Programming*, 20, 1 (1991) 23-53.

[9] K.N. Ganapathy and B.W. Wah, Optimal synthesis of algorithm-specific lower-dimensional processor arrays. *IEEE Trans. on Parallel and Distributed System*, 7 (3) (March 1996) 274-287.

[10] M. Griebl and C. Lengauer, The loop parallilizer LooPo. In Proc. *Sixth Workshop On Compilers For Parallel Computers*, M. Gerndt, Ed. Konferenzen des Forschungszentrums Jülich, vol. 21, (1996) 311-320.

[11] A. Gupta, F.G. Gustavson, M. Joshi, S. Toledo, Design, Implementation, and Evaluation of Banded Linear Solver For Distributed-Memory Parallel Computers. *IBM Research Report*, RC 20481 (6/19/1996).

[12] R. M. Karp, R. E. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. Journal of ACM, 14(3) (1967) 563-590.

[13] H.T. Kung,"Why Systolic Architecture ?", *IEEE Computer*, 15 (1) (1980) 37-46.

[14] L. Lamport. The parallel execution of Do loops. *Communication of ACM*, 17 (2) (1974) 83-93.

[15] B. Louka and M. Tchuenté, Dynamic programming on two-dimensional systolic arrays, *Information Processing Letters*, 29 (2) (September 1988) 97-104

[16] B. Louka and M. Tchuenté, Triangular matrix inversion on systolic arrays, *Parallel Computing*, 14 2 (June 1990), 223-228.

[17] J.W.H. Liu, Computational models and task scheduling for parallel sparse cholesky factorization. *Parallel Computing*, 3 (4) (1986) 327-342.

[18] D.I. Moldovan. On the analysis and synthesis of VLSI algorithms. *IEEE Transactions on Computers*, C-31 11 (1982) 1121-1126.

[19] C. Mongenet, Data compiling for systems of uniform recurrence equations. *Parallel Processing Letters*, 4 (3) (1994) 245-257.

[20] P. Quinton. Automatic synthesis of systolic array from uniform recurrence equations. In Proc. *IEEE 11th Annual International Conderence on Computer Architecture*, Ann Arbor, (1984) 208-214.

[21] P. Quinton and V. V. Dongen. The mapping of linear equations on regular arrays. *J. VLSI Signal Processing* 1(2) (1989) 95-113.

[22] S. V. Rajopadhye. Synthesizing systolic arrays with control signal from recurrence equations. *Distrib. Comput.* 3 (1989) 88-105.

[23] S. V. Rajopadhye and R. M. Fujimoto. Synthesizing systolic arrays from recurrences equations. *Parallel Computing*, 14 (June 1990) 163-189.

[24] T. Risset and Y. Robert. Synthesis of processors arrays for the algebraic path problem: unifying old results and deriving new architectures, *Parallel Processing Letters*, 1 (1) (1991) 19-28.

[25] I. Sakho and M. Tchuente. Methode de conception d'algorithmes paralleles pour reseaux reguliers. *Technique et Science Informatiques*, 8 (1989) 63-72.

[26] W. Shang and J.A.B. Fortes, Time optimal linear schedules for algorithms with uniform dependencies. *IEEE Trans. on Computers*, 40 (1991) 723-742.

[27] C.T. Djamegni, P. Quinton, S. Rajopadhye and T. Risset. Derivation of Systolic Algorithms For the Algebraic Path Problem by Recurrence Transformations. *Parallel Computing*, 26 (11) (2000)1429-1445.

[28] C.T. Djamegni and M. Tchuenté. Scheduling of the DAG Associated with Pipeline Inversion of Triangular Matrices. *Parallel Processing Letters*, 6 (1) (1996) 13-26.

[29] C. T. Djamegni and M. Tchuenté. A New Dynamic Programming Algorithm on two dimensional arrays. *Parallel Processing Letters*, 10 (1) (2000) 15-27.

[30] C.T. Djamegni. Contribution to the Synthesis of Optimal Algorithms for Regular Arrays. *Th èse de Doctorat*, Department of Computer Science, Univ. of Yaoundé I-Cameroun, (1997).

[31] M. Tchuenté. Parallel Computation on regular arrays. Manchester University Press and John Wiley Sons, (1992).

[32] L. Thiele, Resource constrained scheduling of uniform algorithms. *Journal of VLSI Signal Processing*, 10 (3) (August 1995) 295-310.

[33] Y. Wong and J. M. Delosme. Optimization of processor count for systolic arrays. *Research Report*, YALEU/DCS/RR-697 (1989).

[34] Y. Wong and J. M. Delosme. Space-optimal linear processor allocation for systolic arrays synthesis. In Proc. *Sixth Int'l Parallel Processing Symp.*, (1992) 275-282.

[35] Yaov Yaacoby and Peter R. Cappelo. Bounded broadcast in systolic arrays, *Technical Report*, Computer Science Department, University of California, Santa Barbara, TRCS88-13 (April 1998)