Mapping Algorithms onto a Multiple-Chip Data-Driven Array

Bilha Mendelson
IBM Israel - Science & Technology
Matam
Haifa 31905, Israel
bilha@vnet.ibm.com

Israel Koren
Dept. of Electrical and Computer Eng.
University of Massachusetts
Amherst, MA 01003
koren@euler.ecs.umass.edu

Abstract

Data-driven arrays provide high levels of parallelism and pipelining for algorithms with no internal regularity. Most of the methods previously developed for mapping algorithms onto processor arrays assumed an unbounded array (i.e., one in which there will always be a sufficient number of PEs for the mapping). Implementing such an array is not practical. A more practical approach would be to assign the PEs to chips and map the given algorithm onto the new array of chips.

In this paper, we describe a way to directly map algorithms onto a multiple-chip data-driven array, where each chip contains a limited number of processing elements. There are two optimization steps in our mapping. The first is to produce an efficient mapping by minimizing the area (i.e., the number of PEs used) as well as optimizing the performance (pipeline period and latency) for the given algorithm, or finding a trade-off between area and performance. The second is to divide the unbounded array among several chips each containing a bounded number of PEs.

1: Introduction

Computationally demanding problems which do not exhibit high regularity are unsuitable for systolic and wavefront arrays [1]. The data-flow mode of computation seems to be an appropriate approach to follow for this type of problem. We therefore adopt the approach proposed in [4] (and used also in [9] and [8]), where the algorithm is first represented in the form of a data flow graph and then mapped onto a data-driven processor array. The processors in this array execute the operations included in the corresponding nodes (or subsets of nodes) of the data–flow graph, while regular interconnections of these elements serve as edges of the graph. A data-driven processor array is a programmable and homogeneous array which is composed of a mesh of identical processing elements (PEs). The hexagonal topology of the array, shown in this paper, serves only as an example of a regular structure.

A data-driven PE is capable of performing arithmetic, logical, routing and synchronization operations, and can contain more than one operation. In particular, the routing operations can be performed in parallel to the arithmetic and logic operations. In this way, data passing through a PE is not blocked by internal computations. However, the remaining operations assigned to the same PE are executed sequentially according to the arrival of their operands.

A method for mapping a given algorithm onto a data-driven array to achieve a better performance was presented in [7]. This mapping involves assigning every node of the data flow graph (DFG) to an element in a data-driven processor array. Figure 2 shows a mapping

0-8186-3492-8/93 $3.00 © 1993 IEEE

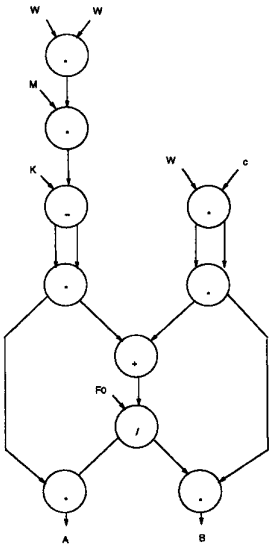


Figure 1. A data flow graph.

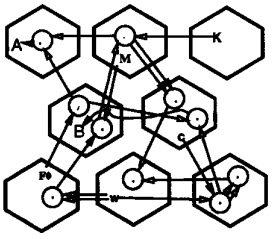


Figure 2. The mapping of the graph in Figure 1 onto a hexagonally connected array.

of the graph in Figure 1 onto a hexagonally connected array. Note that the use of the data-flow graph identifies the implicit parallelism of the computation as described in [4]. There are two optimization steps in this mapping. The first is to produce an efficient mapping by minimizing the area (i.e., the number of PEs used) as well as optimizing the performance (pipeline period and latency) for the given algorithm, or finding a trade-off between area and performance. The second is to divide the unbounded array among several chips each containing a bounded number of PEs.

Most of the methods previously developed for mapping algorithms onto processor arrays assumed an unbounded array (i.e., one in which there will always be a sufficient number of PEs for the mapping). Implementing such an array is not practical because of the unbounded number of PEs it contains. A more practical approach would be to assign the PEs to chips and map the given algorithm onto the new array of chips.

Most designs of very large regular arrays (such as systolic and gate arrays) do not deal with the question of how the elements of the array should be divided into individual chips. There are two main approaches to solving this problem. The first is to enlarge the capacity of a chip, which means designing a large piece of silicon that will contain the original design in one piece (i.e., wafer scale integration [1]). This approach is expensive and necessitates a fault tolerant implementation due to defects from which this technology suffers. A different approach, and the one most commonly used, is to divide the large array into a number of clusters. The restrictions of the available technology (i.e., number of transistors and number of pins) can then satisfied. A commonly used algorithm for this problem is the min-cut algorithm [5, 3, 2] which divides a graph into clusters with a minimum number of connections between them, while preserving the limit on the number of nodes that every cluster can contain.

This paper addresses the problem of how to map the given algorithm onto a multiple-chip data-driven array. Section 2 briefly describes the use of the simulated annealing for mapping DFGs. Section 3 discusses the arrangement of the chip. Section 4 describes the proposed algorithm and several examples illustrating this algorithm are presented. Conclusions and future research are presented in the last section.

2: Simulated Annealing and the Mapping Problem

Simulated annealing is a powerful algorithm for solving combinatorial optimization problems [6]. If the given problem has a regular structure, it is likely that a good heuristic algorithm, which can take advantage of the problem's structure, will yield better results. On the other hand, when there is no such structure it is better to use some variant of the simulated annealing algorithm.

As was shown in [7], the mapping algorithm for data-driven arrays falls into the last category. Hence, we choose this method in order to find a good mapping. To use the simulated annealing algorithm for the mapping process, we have to define some basic parameters. These include the initial configuration, acceptable moves and the cost function. The initial configuration is some feasible mapping where every DFG node is assigned to a PE (the simple mapping procedure presented in [4] can be used to generate such an initial configuration).

Acceptable moves that change a configuration are:

- moving a node from one PE to another.
- exchanging nodes between PEs.

The first move allows the nodes to move from one PE to another PE which has not used all its capacity, while the purpose of the second move is to allow reassignment of nodes even for PEs that have used all of their capacity.

For several applications there is a need for area minimization or performance maximization. But in most of the applications, there is a need to combine these two. Therefore, three criteria for evaluating a mapping were suggested in [7]:

Area - number of PEs utilized by the mapped DFG

Pipeline period - mean time between successive results

Latency - time elapsed from entering the input operands until the output is produced

A good mapping is one that optimizes all three criteria according to the weight coefficients that are supplied to the simulated annealing algorithm. The cost function, which is used to evaluate the new configuration, is given by

$$\lambda_a A + \lambda_p P + \lambda_l L$$

where A is the area (number of PEs) being used, P is the estimated pipeline period, L is the estimated latency and λ_i's are weight coefficients ($\Sigma\lambda_i = 1$, $i = a, p, l$).

3: Chip Description

We must now figure out how to divide the data-driven PEs into chips in order to implement the proposed hexagonal array. There are several parameters that need to be considered

in order to determine the number of PEs that can be packaged into a single chip, such as the number of transistors a chip can contain and the number of pins it can have. The amount of transistors that can be included in a single chip has increased dramatically in the last few years (chips with 2 to 3 million transistors are currently being designed). On the other hand, the number of pins a chip can have is a more restrictive limit. Currently, the limit on the number of pins is around several hundred pins per chip. Figure 3 shows three suggestions for chip arrangement (i.e., number of PEs in a single chip). Figure 3(a), Figure 3(b) and Figure 3(c) illustrate the packaging of 3, 4 and 6 PEs on a single chip, respectively. The PEs are connected to their neighbors through double links like those in the homogeneous array regardless of whether they are inter-chip or intra-chip links.

The number of pins of a chip containing n PEs can be computed according to the following recursive equation:

$$\begin{aligned} P_{in}(1) &= P_{one} \\ P_{in}(n) &= P_{in}(n-1) + (P_{one} - 2 \cdot E \cdot IN) \end{aligned}$$

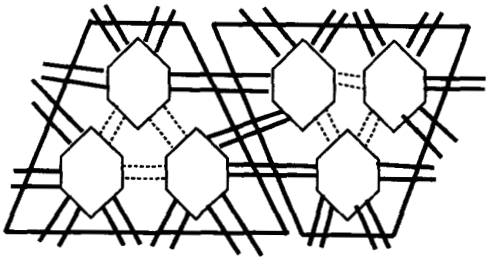
where n is the number of PEs in the chip, P_{in} is the number of edges of a PE, E is the number of edges connected to a neighbor and IN is the number of inner neighbors of the PE that was added. Note that the number of the edges that must be deducted is twice the number of edges connected to the neighbors since they are removed from both PEs.

The ratio of the number of pins to the number of PEs in a chip must be examined in order to select a suitable packaging. Figure 4 shows this ratio as a function of the number of PEs accommodated in a single chip. From this figure we can see that the ratio decreases rapidly for small numbers of PEs. Therefore, packaging less than four PEs in a single chip is not recommended. The number of pins that a chip can have is limited and it should be considered. We assume that several hundred pins on a chip will be available in the near future. Six PEs that comprise a single chip require $36 \cdot 16 = 576$ pins for data communication.

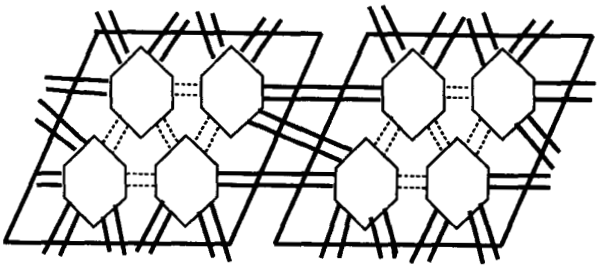
We have seen that the number of connections outside the chip is the major limit for our proposed multiple-chip array. We can increase the number of PEs that are packaged in the same chip by limiting their degree of communication outside the chip. PEs that are facing the chip sides will not have their full degree of connectivity. Figure 5 demonstrates a suggestion for such a chip. The solid lines and dashed lines represent intra-chip and inter-chip communication lines, respectively. We can see that the PEs that are on the rectangular edge have a limited number of links for communication outside the chip. In this design we have chosen to limit the number of pins to $38 \cdot 16 = 608$.

Note that the connectivity between neighboring PEs inside the chip can be increased by adding registers and links for routing data from one PE to another PE. These additional links may reduce the number of long routing links between the nodes of the DFG. The additional internal links may also release some links that are connected to the edge of the chip.

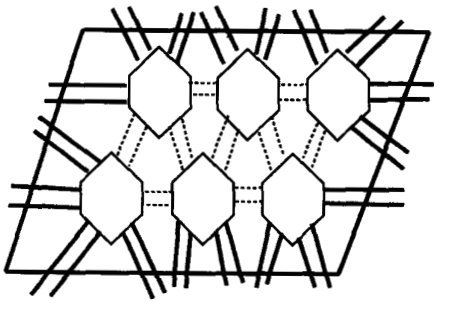
In this design we have conserved the structure of the array. However, there is a difference in the communication delay associated with each link. After being divided into chips, the data-driven array changed from a homogeneous array to a non-homogeneous one. The next section will explain how the same simulated annealing algorithm used for homogeneous arrays can be modified for this new multiple-chip data-driven array.



(a)



(b)



(c)

Figure 3. Chip arrangements: (a) 3 PEs in a chip (b) 4 PEs in a chip (c) 6 PEs in a chip.

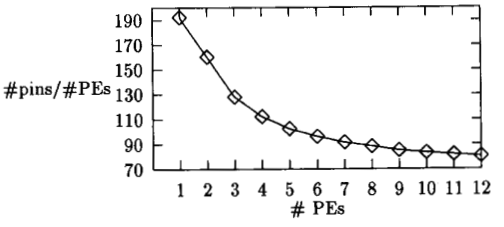


Figure 4. Ratio of number of pins to the number of PEs in a single chip.

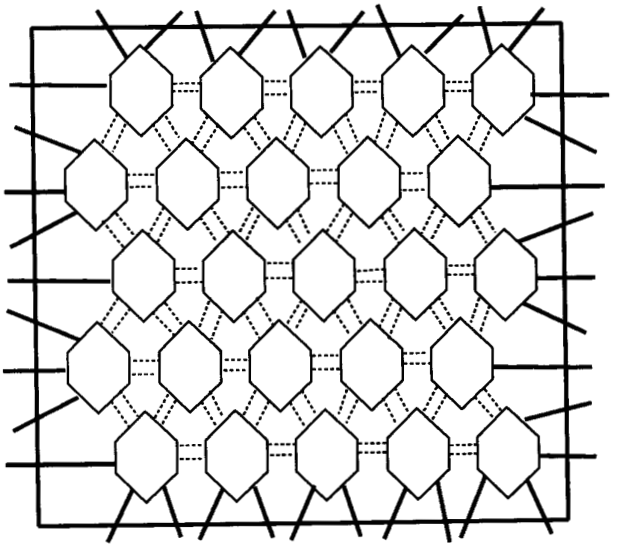


Figure 5. A chip with a limited degree of inter-chip communication.

4: Mapping onto Multiple-Chip Data-Driven Arrays

By dividing the identical PEs into chips the data-driven array becomes non-homogeneous. Therefore, the method for mapping algorithms onto homogeneous data-flow arrays must be modified. The intuitive solution to the problem of mapping onto multiple-chip data-driven arrays is to divide the mapping into two steps, first mapping the algorithm onto the homogeneous array, and only then trying to minimize the number of chips used. In this process, two different heuristic algorithms need to be developed, one for mapping onto the homogeneous array, and the other for slicing it into groups of PEs to accommodate the proposed chip. Because of the increase in the inter-chip communication delay, several disadvantages arise. The first mapping tries to maximize the performance assuming a homogeneous array (i.e., there is a uniform delay on the links among the neighboring PEs). By slicing the array and dividing it into individual chips, the number of routing edges and their delays are increased and thus, the execution time of the mapped algorithm increases. Our solution is to combine the two steps and modify the simulated annealing algorithm for mapping algorithms onto multiple-chip data-driven arrays. In what follows, we present the modifications needed in the mapping algorithm.

When applying the modified mapping algorithm, the basic structure of the array is preserved, changing only the links of the PEs that are on the boundaries. The array links are no longer identical. In a multiple-chip data-driven array there are three types of edges: on-chip edges, which connect PEs on the same chip, inter-chip edges, which connect PEs in different chips, and disconnected edges, which have only one PE connected to them. We will assign a weight to each edge of the data-driven array to reflect the associated communication cost. There is a substantial difference between on-chip communication speed and inter-chip communication speed. Hence, the ratio between the weights assigned to on-chip edges and inter-chip edges should be determined by the ratio between speeds of on-chip and inter-chip communication. A value of ∞ is assigned to disconnected edges.

The simulated annealing algorithm presented in [7] assumes a single type of link and the delay on the links has therefore been neglected. In our case, there is a need to update the simulated annealing cost function in order to take these different types of links into account. The nodes that are assigned to the same PE are executed sequentially and affect the pipeline period. In data-driven computation, a node cannot be operated until all its outputs have been consumed. Since not all the output links have equal communication delays, the pipeline period becomes

$$P = \max(\max_{N}(EX(node_j) + D_j), \max_{M}(\sum_{i=1}^{n_i} EX(node_i)))$$

where N is the number of nodes in the DFG of the given algorithm, M is the number of PEs utilized in the mapping, D_j is the largest delay of the output from $node_j$, and n_i is the number of nodes that were assigned to the same PE.

We also modified the calculation of the latency to include the additional delay which affects it. We differentiate between routing and data manipulation inside the chip and outside it and add the time it takes for the data to move from one PE to its neighbor. We will demonstrate the modification in the simulated annealing mapping algorithm through examples with chips containing six PEs.

When integrating several PEs on a single chip, the communication speed (between neighboring PEs) can increase by a factor of four. The communication delay on the links between neighboring PEs inside the chip is one clock cycle, and the delay outside the chip is four times larger than that.

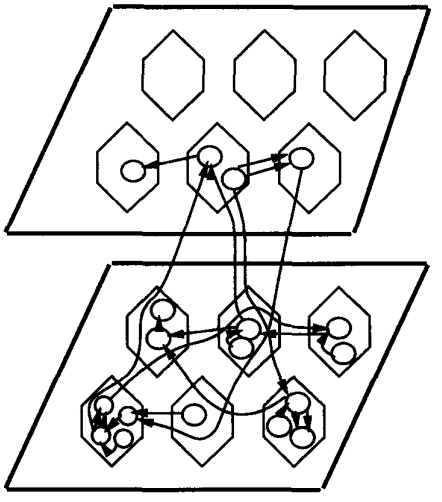


Figure 6. Mapping using the modified simulated annealing process.

Figure 6 shows the result of mapping the DFG in Figure 1 after performing the modified simulated annealing process. In this example the goal was to minimize the area as well as the pipeline period. In this mapping the number of PEs decreased by 43.33% from the initial mapping, which is a substantial improvement over the initial configuration and the pipeline period is the best that can be achieved. We can see that two chips are needed for this mapping.

In order to analyze the efficiency of our method, we need to see what additional overhead was introduced by the modifications we made in the cost function of the simulated annealing. Figure 7 compares the average number of iterations needed by the simulated annealing to obtain a mapping of the algorithm in Figure 1. We can see that the modifications we added to the cost function had actually decreased the number of iterations. The reason is that since the array is no longer homogeneous, the required search time to find the minimum pipeline period is reduced.

Figure 8 describes the average area, pipeline period and latency achieved when mapping the example in Figure 1 onto a homogeneous data-driven array. Figure 8 illustrates the same parameters when mapping the same example onto the multiple-chip data-driven array. We used simulated annealing with almost equal weight coefficients $((\lambda_a, \lambda_p, \lambda_l) = (0.4, 0.3, 0.3))$, and with $(\lambda_a, \lambda_p, \lambda_l) = (0.5, 0.5, 0)$ as a function of the cooling rate.

We can observe that for a cooling rate of 0.95 (which was recommended in [7]) the same number of PEs is used. The performance improves as the cooling rate increases and therefore, the cooling rate of 0.95 is also recommended here. The difference between the pipeline period and latency data in the two figures is due to the fact that links weights are introduced in the multiple-chip array. However, the same behavior can be seen in both cases. An interesting phenomenon can be seen in the latency graphs. In the unbounded array, if area minimization was of interest, the latency did not decrease as the area decreased. In the multiple-chip array cases, there is a correlation between the area and the latency.

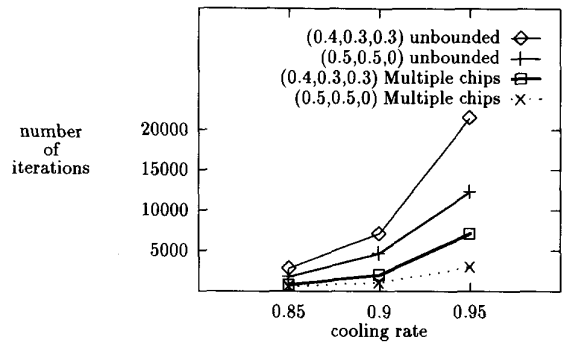


Figure 7. Average number of iterations needed by the simulated annealing to obtain a mapping of the algorithm in Figure 1 for the unbounded array and the multiple-chip data-driven array (for two sets of values for $(\lambda_a, \lambda_p, \lambda_l)$).

In order to illustrate the mapping method for the multiple-chip data-driven array with limited chip connectivity, we reduced the inter-chip communication capacity of the array shown in Figure 3(c) to a single link (instead of a double link) for the inter-chip communication, as shown in Figure 9. There is no significant difference in the area and performance of the first example when mapped either onto the unbounded array or onto the multiple-chip array, due to its size. We have also applied the simulated annealing algorithm to the second example in [7] (shown in Figure 10). Figure 11 demonstrates a mapping of the example shown in Figure 10 onto the chip array shown in Figure 9. The average number of chips this example needs increased from 6.1 chips on the multi-chip array shown in Figure 3(c) to 8.7 on the limited one shown in Figure 9.

5: Conclusions

A method for mapping algorithms onto multiple-chip data-driven arrays has been presented. This is achieved by assigning weights to the data-driven array connections. The method can be used on arrays that limit their out-degree and therefore remain homogeneous in regard to their topology, but have different delays on the links. It can also be applied when the number of outgoing links of PEs at the chip boundary is reduced.

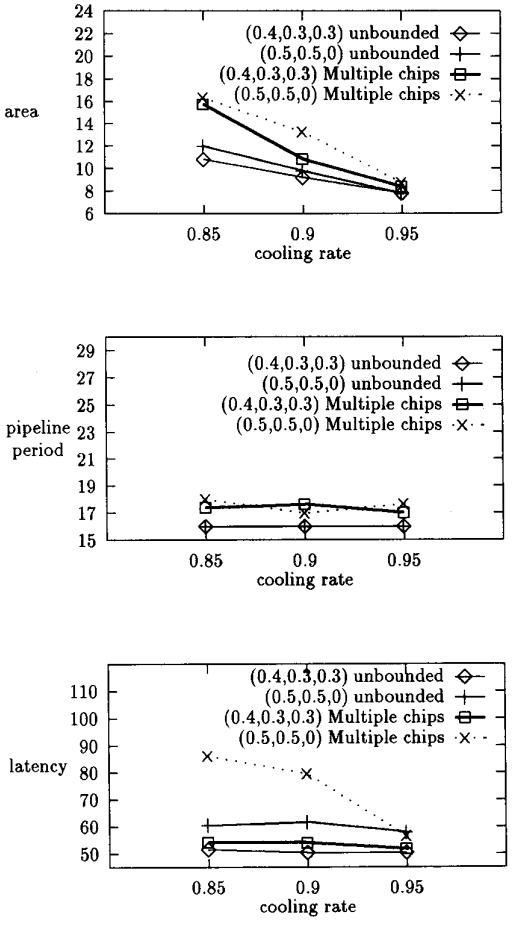


Figure 8. Average area and performance achieved when mapping the example in Figure 1 onto a homogeneous unbounded array and multiple-chip data-driven array.

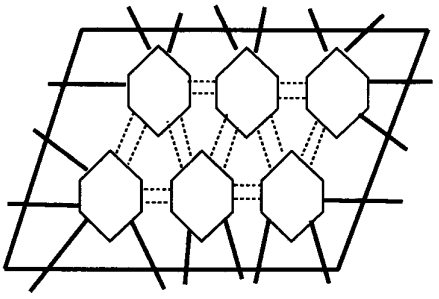


Figure 9. Chip arrangement of 6 PEs in a chip with reduced connectivity.

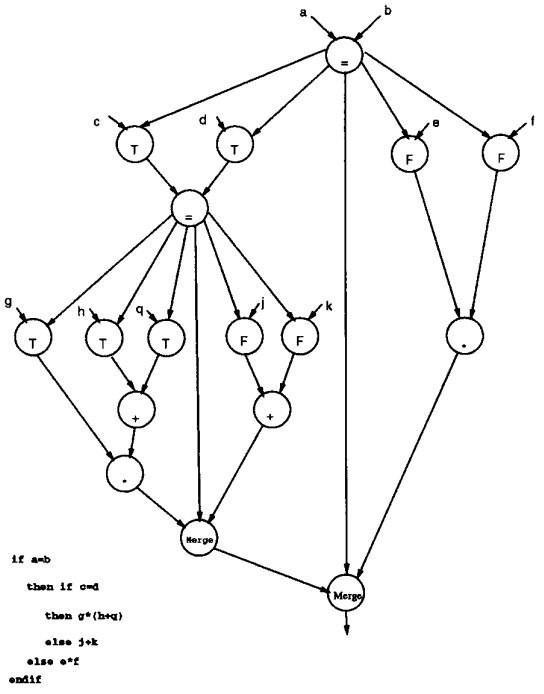


Figure 10. Nested if-then-else expression and its DFG representation.

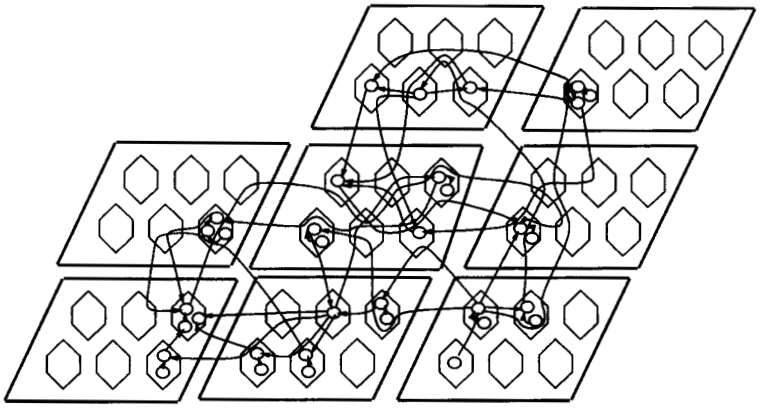


Figure 11. Mapping of the example in Figure 10 onto the reduced connectivity chip array.

References

[1] G. Saucier A. Baubekeur, J. Patry and J. Trilhe. Configuaring a wafer-scale two-dimensional array of single-bit processors. *IEEE Computer*, 25(4):29–39, April 1992.

[2] David S. Johnson et al. Optimization by simulated annealing: An experimental evaluation, part I. *Oper. Res.*, 37:865–892, Nov.-Dec. 1989.

[3] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell system Technical Journal*, 49:291–307, February 1970.

[4] I. Koren, B. Mendelson, I. Peled, and G. M. Silberman. A data-driven VLSI array for arbitrary algorithms. *IEEE Computer*, pages 30–43, October 1988.

[5] B. Krishnamurthy. An improved min-cut algorithm for partitioning vlsi networks. *IEEE Trans. on Computers*, C-33(5):438–446, May 1984.

[6] P.J.M. Van Laarhoven and E.H.L Aats. *Simulated Annealing: Theory & Applications.* D.Reidel Publishing Company, 1987.

[7] B. Mendelson and I. Koren. Using simulated annealing for mapping algorithms onto data driven arrays. In *Proc. of 1991 International Conf. on Parallel Processing*, Aug. 1991.

[8] B. Robic, P. Kolbezen, and J. Silc. Area optimization of dataflow-graph mappings. *Parallel Computing*, (18), 1992.

[9] J. Serot, G. Quenot, and B. Zavidovique. A functional data-flow architecture dedicated to real-time image processing. In *IFIP 10.3 Working Conference on Architectures and Compilation Techniques for Fine and Medium Grain Parallelism*, 1992.