Reiner W. Hartenstein, Jürgen Becker, Michael Herz, et al.: A Synthesis System for Bus-based Wavefront Array Architectures; Proceedings of ASAP 96 Application Specific Array Processors, Chicago, USA, August 1996

A Synthesis System for Bus-based Wavefront Array Architectures

Reiner W. Hartenstein, Jürgen Becker, Michael Herz, Rainer Kress, Ulrich Nageldinger

University of Kaiserslautern
Erwin-Schrödinger-Straße, D-67663 Kaiserslautern, Germany
Fax: ++49 631 205 2640, email: abakus@informatik.uni-kl.de

Abstract

A datapath synthesis system (DPSS) for a bus-based wavefront array architecture, called rDPA (reconfigurable datapath architecture), is presented. An internal data bus to the array simplifies the access of the processing elements for data manipulations. The DPSS allows automatic mapping of high level datapath structures onto the rDPA without manual interaction. Optimization techniques are sketched. The rDPA is scalable to arbitrarily large arrays and reconfigurable to be adaptable to the computational problem. Fine grained parallelism is achieved by using simple reconfigurable processing elements which are called datapath units (DPUs). The rDPA can be used as a reconfigurable ALU in transport-triggered architectures as well as for rapid prototyping of high speed datapaths.

1. Introduction

Many computation-intensive algorithms take too much execution time, even on a well-equipped modern workstation. This opens a market for hardware accelerators of all kinds. Custom configurable accelerators have the advantage to be adaptable to the computational problem. Such an accelerator should be scalable. This means that it should be extensible to various sizes depending on the computational needs. Wavefront arrays suit well for accelerators since their data-driven computation is self-timed and pipelining can easily be performed [5]. The highly parallel I/O requirements of the wavefront array leads to the idea to integrate a bus as an auxiliary structure into the array. In the following the architecture is called reconfigurable datapath architecture (rDPA). In this architecture the decentralized control of a systolic array is combined with the centralized control for I/O data operations.

A controller allows to use the rDPA as a reconfigurable ALU (rALU). This rALU is intended for the parallel and pipelined evaluation of compound operators consisting of complete expressions and statement sequences. In scientific computations such a statement sequences or statement block usually occur in loops. The loop is controlled by multiple *for*-statements evaluating the statement block several times (figure 1). The control of these loops can be performed by a data sequencer. A data sequencer computes generic address sequences from a few parameters. Such a data sequencer is used e. g. in the Xputer [2]. Xputers are especially designed to reduce the von-Neumann bottleneck of repetitive decoding and interpreting address and data computations. Xputers require memory accesses for data only, whereas von-Neumann computers require memory accesses for instructions also. In contrast to von Neumann machines an Xputer architecture strongly supports the concept of the “soft ALU” (rALU).

For a high user acceptance, a synthesis system should be available which is able to map the compound operator onto the rDPA wavefront array without manual interaction. Such a synthesis system, the datapath synthesis system (DPSS) is presented in this paper. The DPSS is integrated into the Xputer application development environment [4] using the Xputer as universal embedded accelerator to state-

```
for (j = 0, …, …)                          (1)
   for (i = 0, …, …) {                     (2)
      /* start of statement block */       (3)
      y[i] = a + b[i] * c[i-1] …;          (4)
      if (a < 3) …;                        (5)
      …                                    (6)
      /* end of statement block */         (7)
   }                                       (8)
```

Figure 1. Statement block (compound operator) in two *for*-loops

of-the-art workstations. The following section sketches the reconfigurable datapath architecture. Section 3 explains how this architecture can be used as a data-driven rALU by describing the rALU controller interface. The datapath synthesis system is introduced in section 4.

2. Reconfigurable Datapath Architecture

The reconfigurable datapath architecture (rDPA) consists of reconfigurable processing elements, which we call datapath units (DPUs). Each rDPA chip contains four by four DPUs. Connecting an array of two by four rDPA chips on a PCB board, an array of 128 DPUs can be realized (figure 2).

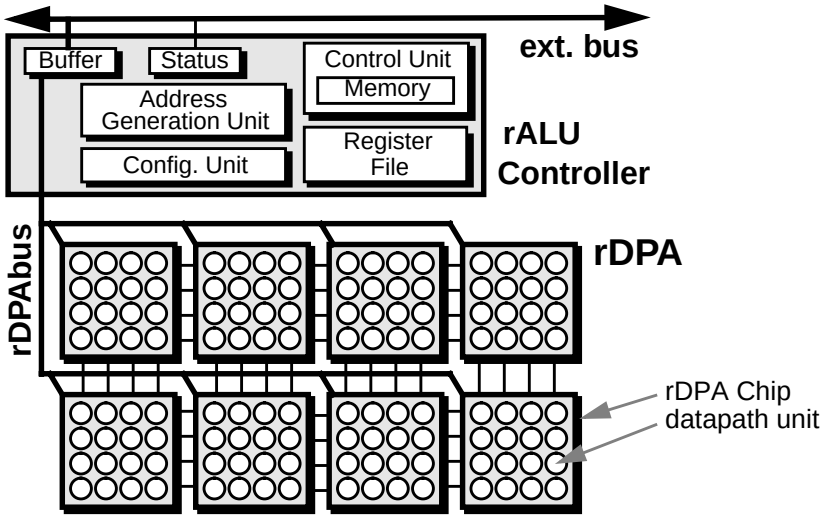

Figure 2. The reconfigurable datapath architecture (rDPA) with the programmable rALU controller

Datapath unit architecture. The rDPA consists of a regular array of identical datapath units (DPUs). Each DPU has two or three input and two or three output registers depending on the routing structure of the rDPA chip (figure 3). The operation of the DPUs is data-driven. This means that the operation will be evaluated when all required operands are available. An extensible repertory of operators for each DPU is provided by the datapath synthesis system from a DPU library. This repertory includes the operators of the programming language C. The architecture of the DPUs consists of a datapath including an ALU and a microprogrammable control unit. Operators such as addition, subtraction or logical operators can be evaluated directly, and larger operators like multiplication or division are implemented by a microprogram. New operators can be added with the aid of a microassembler.

Routing architecture. The rDPA provides two interconnection levels: short parallel connections, and an internal rDPAbus. The local interconnect of the rDPA is implemented as a mesh or as a hexago-

Figure 3. Possible routing structures of the rDPA chip a) mesh, b) hexagonal array

nal array. A mesh compared to other array structures is best suited regarding I/O requirements and scalability. For a large number of DPUs per chip, the number of I/O interconnections and the required pins of the hexagonal array is as twice as much as the ones of the mesh. To reduce the number of input and output pins, a chip connection with partly serial interconnections (4-bit) is used for data transfer between neighbouring DPUs on different chips. Internally the full datapath width is used. For the user and the software this chip connection is completely transparent.

number of DPUs	organization	number of necessary I/O interconnect	
		mesh	hexagonal array
n m	n x m	2 (n + m)	4 (n + m) - 2

Table 1. Number of necessary I/O connections at the array boundary

Both structures allow to execute systolic algorithms efficiently, since these algorithms mainly use the local interconnect. The hexagonal array suits for a larger class of algorithms. Bidirectional communication means that the direction of the communication is chosen at configuration time. At run time, the path is unidirectional. Bidirectional communication is more flexible in implementing compound operators, but additional chip area is required for bidirectional chip connections and the DPUs.

The rDPAbus provides a connection to each datapath unit (DPU). In the rDPA it is used for I/O of operands from outside into the array, and for propagation of interim results to other DPUs far away. A scheduling can determine an efficient usage of this dynamic network. There are several alternatives: A single rDPAbus is sufficient to connect all datapath units. Two buses can speed up I/O operations, especially when each DPU has access to both buses. A cheaper solution is to connect every second rDPA chip to a separate bus. The rALU controller is then in charge to supervise both buses. The data transfers are synchronized data-driven by a handshake like the internal communications. With the proposed routing architecture, the rDPA can be expanded also across printed circuit board boundaries, e.g. with connectors and flexible cable. Furthermore it is possible to build a torus structure.

Prototype. The current prototype of the rDPA chip consists of an three by three array of DPUs. They are interconnected with a unidirectional mesh. A single rDPAbus is used. An earlier version of the wavefront array structure of the rDPA was described in [3].

Configuration. The array is scalable by using several chips of the same type. The DPU address for register addressing of the bus is configured at the beginning. The communication structure allows dynamic in-circuit reconfiguration of the rDPA. This implies partial reconfigurability during run-time. The rDPA can be configured from any chip connection at the array boundary. A single chip connection is sufficient for the complete array, but multiple ports can be used for speedup. Using the rALU controller, also parallel configuration for bus-oriented systems is possible. The configuration is data-driven, and therefore special timing does not have to be considered.

3. The rALU Controller

Together with the rDPA, a programmable rALU controller is provided. Both, the rDPA and the rALU controller form a data-driven reconfigurable ALU (rALU). The rALU controller consists of a rDPA control unit, a register file and an address generation unit for addressing the DPUs (figure 2).

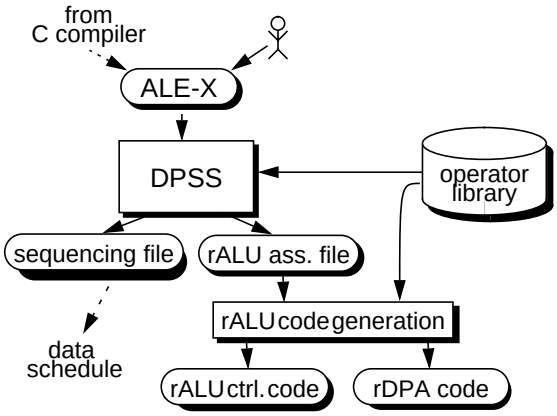

Figure 4. Overview on the datapath synthesis system (DPSS)

Register file. It is useful for optimizing memory cycles, e.g. when one data word of a statement will be used later on in another statement. Then the data word does not have to be read again from the main memory. Currently the register file has 64 34-bit registers (32-bit data, 2-bit status).

Address generation unit. It delivers the address for the DPU registers before each data is written into the rDPA over the bus.

rDPA control unit. It holds a program to control the different parts of the data-driven rALU. The instruction set consists of instructions for loading data into the rDPA array to a special DPU from the external units, for receiving data from a specific DPU, or branches on a special control signal from the host. The control program is loaded during configuration time.

4. The Datapath Synthesis System

The datapath synthesis system (DPSS) allows to map statements from a high level language description onto the rDPA. The statements may contain arithmetic or logic expressions, conditions, and loops, which evaluate iterative computations on a small amount of input data. The input language of the DPSS is called ALE-X (arithmetic and logic expressions for Xputers). It can be edited manually, or it can be generated from the X-C compiler [8], which is also integrated into the Xputer application development environment [4].

The task of configuring the rDPA is carried out in the following phases: logic optimization and technology mapping, placement and routing, and data scheduling. Partitioning of the statements onto the different rDPA chips is not necessary since the array of rDPA chips appears to be a single large array of DPUs with transparent chip boundaries. The DPSS produces an assembler file whereof the configuration files for the rALU and the sequencing file are generated. The sequence is called data schedule. The rALU code generation produces a control file for programming the rALU controller and an rDPA code file for the configuration of the reconfigurable datapath architecture. An overview on the datapath synthesis system is given in figure 4.

4.1 The ALE-X Programming Language

To simplify the programming by hand, the ALE-X programming language should be easy to read, learn, and understand. For this reason, the language is strongly oriented on the concepts of the programming language C. A rALU subnet description contains a part where the interface of the circuit is specified. This means the input and output (I/O) ports, namely the input and output variables as well as their data format is expressed. Another part describes the compound operator consisting of expressions, condition statements (*if-else*) and loop statements such as *while-loops* and *do-while*-loops.

The ALE-X Hardware File. This file allows to use the datapath synthesis system in a very flexible way. If the hardware of the rALU changes, e.g. new operators are available, only a new hardware file has to be specified. The limits of the hardware resources (e. g. number of datapath units available in the rALU subnet) are used to check a given ALE-X file if it can be implemented on the current available hardware. The data types and the available operators are also specified. All operators are listed with their delay times. All listed operators must be accessible in the operator repertory of the assembler for the code generation. The user can extend the available operator library by own functions.

4.2 Logic Optimization and Technology Mapping

The condition statements are converted into single assignment code. The same is done for the loop conditions. The loop header is controlled by the rALU controller. Loops and sequences of assignments are considered as basic blocks. Directed acyclic graphs (DAGs) are constructed from the basic blocks. Herewith common subexpressions, identical assignments, local variables, and dead code are removed. Further constant folding and reduction in strength is used. Unary operators are combined with the next operator if the operator library provides this newly merged operator. This step reduces the number of required DPUs in the rDPA array. Further, parallelism of single expressions is increased by tree-height reduction [6]. This is done to a level where an expression can be placed with at most one routing operator per local interconnection. A simple algorithm is performed which uses the commutativity and the associativity of some operators. If expressions can be vectorized, it is sufficient to implement one of these expressions, thus saving area. The required results can be computed by pipelining this single expression.

4.3 Placement and Routing

A poor placement degrades the performance since some internal variables have to be routed via an rDPAbus. During that time this bus is blocked for other data operations. A simulated annealing algorithm is chosen which gives better results than a simple constructive cluster growth algorithm, especially when the rDPA is connected as a torus. Different torus structures for connecting the border of the

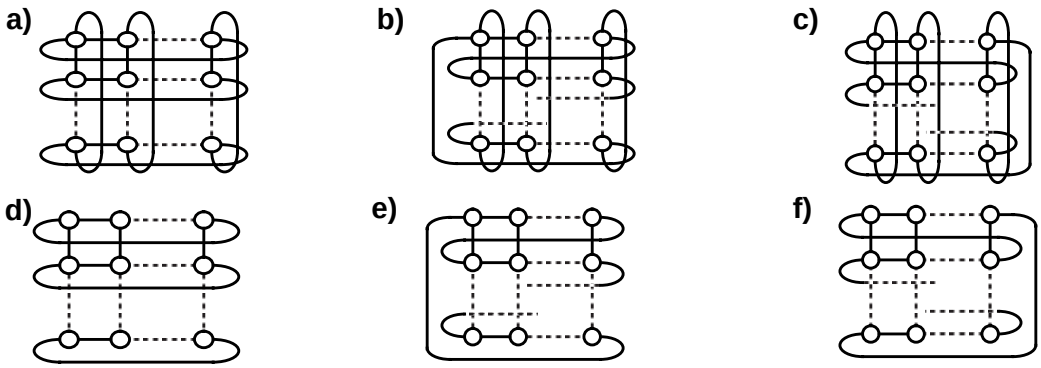

Figure 5. Overview on diffferent Torus structures used by DPSS

rDPA can be additionally considered (see figure 4). The simulated annealing algorithm belongs to probabilistic algorithms which improve iteratively [7]. One additional routing operation per local connection is considered in each iteration step by the simulated annealing algorithm. Thus the routing is considered during the placement. Unlike other simulated annealing algorithms, the minimum placement is always saved in each iteration step. The cost function of the annealing algorithm considers the chip boundaries, the required routing operators and with a high cost the connections via the rDPAbus. Figure 6 shows the implemented algorithm.

First, an initial placement for the simulated annealing process is computed by placing the operators randomly onto the rDPA. Then the *COST*-function gives the complete cost of this placement. The cool-

```
/* Algorithm Simulated Annealing for the DPSS */
placement = INITIAL_PLACEMENT();
actual_cost = COST(placement);
minimum_cost = actual_cost;
for (temp = START_TEMP; temp > FINAL_TEMP && actual_cost != 0;
                                                SCHEDULE(temp)) {
  /* inner loop */
  for (i = 1; i <= MAX_ITERATION(temp) && actual_cost != 0; i++) {
    /* select two positions for exchange */
    position1 = SELECT_POSITION();
    position2 = SELECT_POSITION(position1, temp);
    /* change positions and compute cost that results from change */
    delta_cost = EXCHANGE(position1, position2);
    /* accept new placement if … */
    if (delta_cost < 0 || RANDOM(0, 1) < exp (-delta_cost / temp )) {
      /* new minimum cost, retain minimal placement */
      if ( actual_cost < minimum_cost ) {
        minimum_placement = placement;
        minimum_cost = actual_cost;
      }}
    else {
      /* return to previous placement */
      EXCHANGE(position1, position2);
    }}}
placement = minimum_placement;
```

Figure 6. Simulated annealing algorithm used

ing schedule is controlled by a linear function f(temp) = 0.95 * temp. Two positions for exchange are searched by the *SELECT_POSITION()*-function. Without argument, this function gives a random position of an operator in the rDPA array. With arguments, the function returns a random position of an operator in the neighbourhood of the position of the first argument. The second argument determines the range of this neighbourhood. A high value for the second argument allows to search for the new position in the whole array, a low argument allows to search only in a small neighbourhood. Usually, the temperature of the cooling schedule is used for this argument. Further, the number of iterations of the inner loop is decreased with the temperature. This can be done because only a local neighbourhood for exchange is considered at low temperatures. The *EXCHANGE*-function exchanges two positions of the current placement and returns the difference of the costs between the old and new placement. Only the cost increase or decrease due to this exchange is computed. The rest of the operators in the rDPA array are not considered for reasons of speed of the implementation. The *RANDOM()*-function returns a random number in the range between the first and the second argument. An example of an expression statement sequence mapped onto an rDPA with unidirectional mesh connections is given in figure 7.

The annealing algorithm is able to consider up to six bidirectional local connections per DPU in hexagonal array structures (see figure 3). Additionally torus strucutures (figure 4) can be selected, also one for the hexagonal array strucuture.

Experimental results, obtained by analyzing several placements of different applications, has shown, that using bidirectional instead of unidirectional local connections improved the performance/area trade-offs by 20% - 30%. An addition improvement of approximately 10% was caused by using one of the torus structures. But different variations of torus structures as well as the use of hexagonal array structures resulted only in slight quality differences of the placements.

4.4 Data Scheduling

Scheduling determines a precise start time of each operation. The start time must satisfy the dependencies between the operations which limit the amount of parallelization. Since scheduling determines the concurrency of the resulting implementation, it affects its performance.

The placement and routing step of the design implementation takes care, that for each operation, a single datapath unit (DPU) is available. The local interconnect between the DPUs is responsible for transferring intermediate results to the following operation. That means, the arithmetic and logic operations are not constraint. Two other resources are constraint: the rDPAbus and the external bus providing the data from the main memory.

- One or several external buses provide a regular data stream for the rDPA. Each bus is limited to a single data operation per time. The current prototype uses only a single external bus due to a single data memory.
- The number of rDPAbuses is also limited, and their connections to the different rDPA chips have to be known. An rDPAbus is limited to a single data operation per time, too, though as a local bus, it can operate faster than the external bus. The current prototype uses a single rDPAbus since it provides enough speed to serve the external bus.

Thus, constraints can be classified into two major groups: *interface constraints* and *implementation constraints*. Interface constraints are additional specifications to ensure that the circuits can be embedded in a given environment. In our case, these are the timing constraints of the data stream via the external bus. Considering the complete rALU, the I/O constraints of the rDPAbus can be seen as implementation constraints. The rDPAbus is used for internal wide range data transport via the register file, and for the I/O data operations. These transfers to the register file and to the buffer register can be considered as data operations in the schedule. The external bus adds timing constraints to these operations. Further, the sequence of the data words via the external bus has to be the same as the sequence of data words from the buffer register to the rDPA or vice versa. But in-between this sequence via the rDPAbus, getting and putting data words to the register file is allowed. Due to the data-driven synchronization concept of the rALU, the time steps for the scheduling algorithm are very short and most operations use multiple cycles for evaluation. The time unit should be the greatest common divisor of the delay times of all used operations. The scheduling algorithm allows to specify several external buses as well as several rDPAbuses. In the following the examples are shown for the current prototype.

For a deadlock free schedule and an optimal sequence of I/O operations, following strategy is used:

- Perform a resource-constraint scheduling of the *sequencing graph*, including the *get* and *put* operations for I/O of data, but without considering the timing restrictions of the external data stream.
- Since the sequence of operations is fixed now after the first scheduling, the timing constraints of the external data stream are added.
- Perform a timing constraint schedule including the requirements of the external data stream.

```
x1 = x + dx;
u1 = u - 3 * x * u * dx - 3 * y * dx;
y1 = y + u * dx;
c = x1 < a;
```

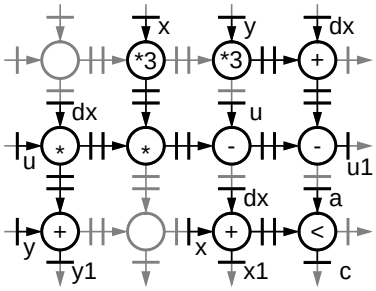

Figure 7. Example of the placement of four expression statements onto the reconfigurable datapath architecture

The goal of this scheduling is to find a deadlock free schedule and an optimal sequence of the I/O operations. First, a minimum-latency resource-constraint scheduling is used. The sequencing graph consists of a set of operations including the I/O operations of the rDPAbus. The vertex set V= $\{v_i; i = 0, 1, 2, \ldots, n\}$ of a sequencing graph $G_S(V, E)$ is in one-to-one correspondence to a set of operations, and the edge set $E = \{(v_i, v_j); i, j = 0, 1, 2, \ldots, n\}$ represent dependencies. The source vertex v_0 and the sink vertex v_n with $n = n_{ops} + 1$ are both no-operations (NOP). The set of execution delays is $D = \{d_i; i = 0, 1, 2, \ldots, n\}$ with $d_0 = d_n = 0$. It is assumed that the delays are data independent and known. Further it is assumed, that the delays are integers and a multiple of the time step.The external bus is not considered at this time. Scheduling is done on basic blocks only. This means, that there is no difference between expression statements and conditions. The same is valid for *while* and *do-while* loops. They are considered by the code generation step. The sequencing graph for the example in figure 7 is listed in figure 8.

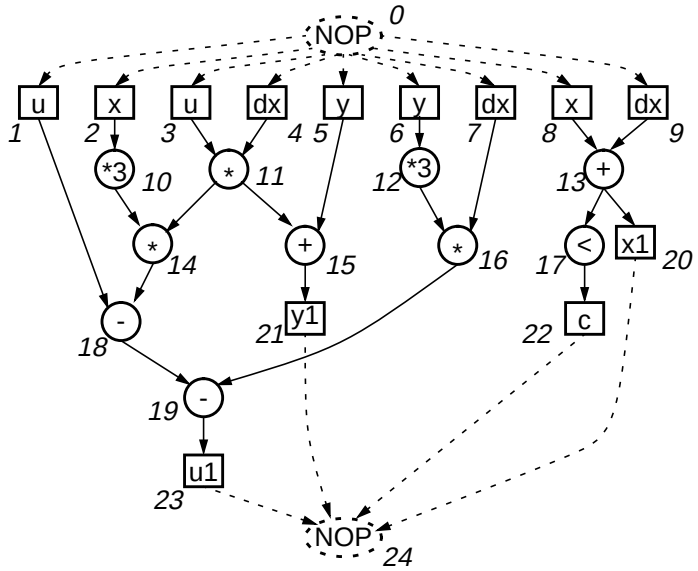

Figure 8. Example of a sequencing graph

For the scheduling algorithm, a list schedule with improvement of the priority list in each iteration step like in the force-directed scheduling is used. The mobility determines the ranking of the operations. The resource constraints are represented by the vector **a**. In our case, only the I/O operations are required for this priority list ($a_{I/O} = 1$), since all other operations are no resource-constraint. The operations whose predecessors have already been scheduled, so that the corresponding operations are completed at time step *l* are called candidate operations $U_{l,k}$, with

$$U_{l,k}=\{v_i \in V : type(v_i)=k \text{ and } t_j+d_j \, l \forall j:(v_j,v_i)\in E\} \qquad (1)$$

for any resource type $k = 1, 2, \ldots, n_{res}$. Figure 9 shows the scheduling algorithm used.

For simplicity it is assumed that each data operation has a delay of three time steps, a multiplication 22, and an ALU operation four. A multiplication with three is implemented as a shift followed by an addition, which requires six time steps in total. A first schedule determines the sequence of data operations via the rDPAbus. Thus the sequence of data words via the external bus can be fixed.

Each multiply used variable is read once from the main memory and then transferred into the rDPA array and concurrently to the register file. This transfer to both locations does not increase the delay time of the I/O operation. Supposing that every five time steps a new variable can be transferred via the external bus gives the minimum timing constraint for the final scheduling.

After the first scheduling the same algorithm as before leads to the final data schedule. Due to the timing restrictions on the external bus, the free time slots on the rDPAbus are used for transfers from the register file to the rDPA registers. Figure 10 shows the final data schedule and the sched-

```
I/O (G_S(V, E), a_I/O) {
  l = 1;
  do {
    /* schedule the I/O operations (k = 1) */
    compute the mobility of the I/O operations _1 = t_1^L - t_1^S
      from ASAP and ALAP schedule;
    determine candidate operations U_l,1;
    schedule the U_l,1 with lowest mobility;
    /* schedule the rest of operations (k   1) */
    for (resource type k = 2; k   n_res; k++) {
      determine candidate operations U_l,k;
      schedule the U_l,k requiring no additional resources;
    }
    l = l +1;
  } while (v_n is not scheduled);
  return ();
}
```

Figure 9. Data scheduling algorithm minimizing latency under resource constraints

ule for both rDPAbus and external bus. The final schedule has a latency of $\lambda = 73$ time steps. From this schedule, the sequencing file can be determined. Further, the schedule is deadlock free.

4.5 Optimization

Optimizations can be made in terms of area or speed. Area improvements can be achieved by pipelining vectorized statements. Speed improvements can be achieved by loop folding or loop unrolling [2] if the statement block in the rDPA is evaluated several times. This requires that these loops are controlled either by an external address generator like in the Xputer [2] or by the host.

Vectorized Statements. Instead of mapping the vectorized statement a few times onto the rDPA, it can be pipelined. Adding timing constraints to force the sequence of vector operations in .

the sequencing graph leads to a constraint sequencing graph which can be scheduled as before.

Loop Folding. This the most popular technique for improving the speed of statement blocks that occur in inner loops, since no additional hardware is required. If the implementation in the rDPA is not I/O bound, the operations can be pipelined across loop boundaries. Dependencies between two iterations are considered by the mapping and the scheduling algorithm. Two iterations are scheduled at once with additional timing constraints of the sequencing graph like in the vectorization example. Loop folding requires a special control signal of the address generator or the host to signal the end of the loop.

Loop Unrolling. With this technique a certain number of loop iterations is unrolled. This action results in a loop with a larger body but with fewer iterations. The larger loop body provides a greater flexibility for improving the processing-speed of the loop. Loop unrolling requires at least twice as much datapath units as without using this technique. Improvement against loop folding can be achieved in designs that are not bound by the I/O. Loop unrolling requires additional control especially if the number of operations is not a multiple of the loop iterations that are unrolled.

5. Conclusions

A datapath synthesis system (DPSS) for a bus-based wavefront array architecture, called rDPA (reconfigurable datapath architecture), has been presented. An internal data bus inside the array simplifies the access of the processing elements for data manipulations. The DPSS allows automatic mapping of high level descriptions onto the rDPA without manual interaction. The required algorithms and experimental results of this synthesis system have been described in detail. Some

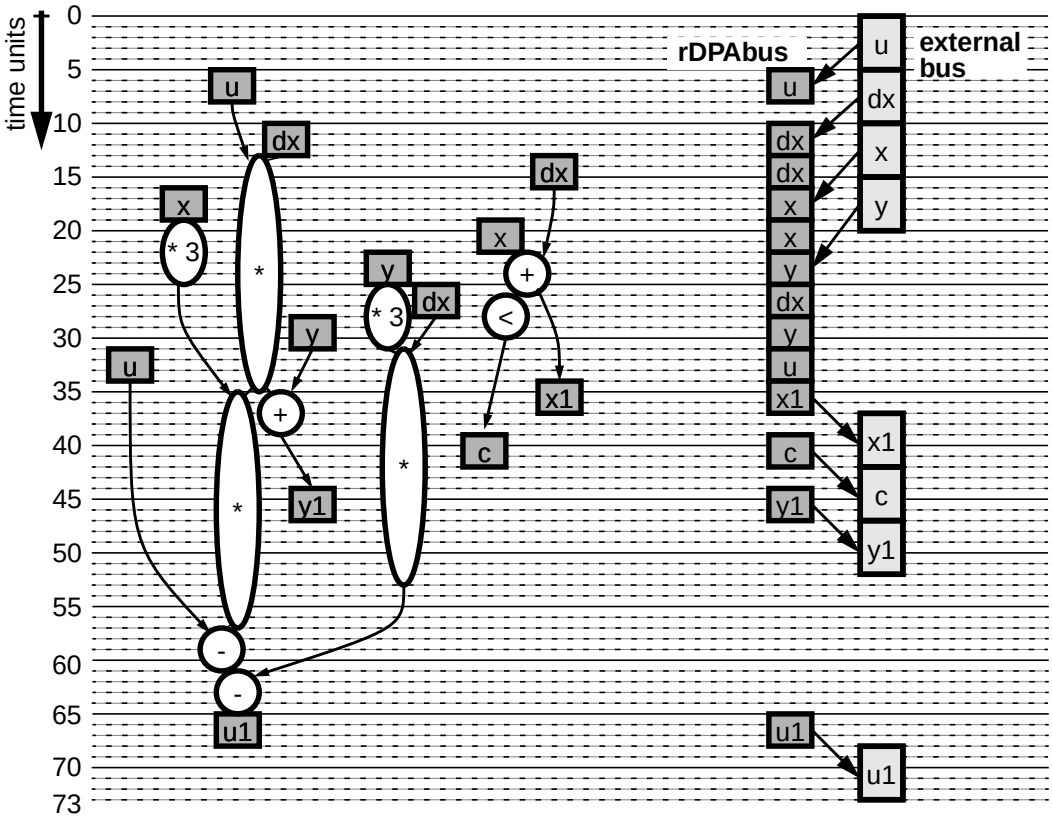

Figure 10. Example of a final schedule for the rDPA taking care of the data stream via the external bus

techniques for optimization such as vectorization for area improvement, and loop folding or loop unrolling for speed improvement have been outlined. The rDPA is scalable to arbitrarily large arrays and reconfigurable to be adaptable to the computational problem. Fine grained parallelism is achieved by using simple reconfigurable processing elements which are called datapath units (DPUs). The rDPA can be used as a reconfigurable ALU as well as for rapid prototyping of high speed datapaths.

References

[1] D. D. Gajski, N. D. Dutt, A. C.-H. Wu, S. Y.-L. Lin: High-Level Synthesis, Introduction to Chip and System Design; Kluwer Academic Publishers, Boston, Dordrecht, London, 1992

[2] R. W. Hartenstein, A. G. Hirschbiel, M. Riedmüller, K. Schmidt, M. Weber: A Novel ASIC Design Approach Based on a New Machine Paradigm; IEEE Journal of Solid-State Circuits, Vol. 26, No. 7, July 1991

[3] R. W. Hartenstein, R. Kress, H. Reinig: A Dynamically Reconfigurable Wavefront Array Architecture for Evaluation of Expressions; Proceedings of the Int. Conference on Application-Specific Array Processors, ASAP'94, San Francisco, IEEE Computer Society Press, Los Alamitos, CA, pp. 404-414, Aug. 1994

[4] R. W. Hartenstein, J. Becker, R. Kress: Two-Level Hardware/Software Partitioning Using CoDe-X; Int. IEEE Symposium on Eng. of Computer Based Systems (ECBS), Friedrichshafen, Germany, March 1996

[5] S. Y. Kung: VLSI Array Processors; Prentice Hall, Englewood Cliffs, New Jersey, 1988

[6] G. De Micheli: Synthesis and Optimization of Digital Circuits; McGraw-Hill, Inc., New York, 1994

[7] N. A. Sherwani: Algorithms for Physical Design Automation; Kluwer Academic Publishers, Boston 1993

[8] K. Schmidt: A Program Partitioning, Restructuring, and Mapping Method for Xputers; Ph.D. Thesis, University of Kaiserslautern, 1994