

Energy Estimation for Piecewise Regular Processor Arrays^{*}

Frank Hannig and Jürgen Teich

University of Paderborn, D-33098 Paderborn, Germany,
{hannig, teich}@date.upb.de,
URL: <http://www-date.upb.de>

Abstract. In this paper, we present a first approach for array-level energy estimation during high-level synthesis when mapping piecewise regular algorithms onto massively parallel full size processor arrays. In-nately, piecewise regular algorithms have some power consumption friendly properties, e.g., they may be mapped onto processor arrays with only local interconnect and memory. In addition to these properties, we show that the chosen mapping has a significant influence on the power consumption. Our energy estimation approach identifies regions with decreased switching activity of functional units' input operands. For these regions with reduced activity, a lower power consumption can be directly obtained from a generated table based model. Experimental results fortify the accuracy and efficiency of our methodology.

1 Introduction

Nowadays, low power has become an important design criterion last but not least due to all the mobile phones and portable computers. These devices have to handle increasingly computational-intensive algorithms like video processing (MPEG4) or other digital signal processing tasks (3G), but on the other hand they are limited in their power budget. The next generation of ULSI chips will allow to implement arrays of hundreds 32-bit micro-processors and more on a single die. Hence, parallelization techniques and compilers will be of utmost importance in order to map computational-intensive algorithms efficiently to these processor arrays.

In this context, our paper deals with the specific problem of estimating the power consumption when mapping a certain class of loop-specified computations called *piecewise regular algorithms* [24] onto a dedicated processor array. This work can be classified to the area of loop parallelization in the polytope model [8, 15].

The rest of the paper is structured as follows. In Section 2, a brief survey of previous work on low power is presented. Section 3 introduces the class of

^{*} Supported in part by the German Science Foundation (DFG) Project SFB 376 “Massively Parallel Computation”.

algorithms we are dealing with. In Section 4, we examine the power consumption of functional units in dependence on their input activity. Afterwards, an energy estimation methodology when mapping regular algorithms to processor arrays is described. The methodology and some results are discussed in Section 5. Future extensions and concluding remarks are presented in Section 6.

2 Related Work

A lot of previous work in the area of low power design during high-level synthesis has dealt with the issue of power estimation. Various methodologies for generating accurate models for datapath power consumption were presented.

In general these power estimation techniques can be divided into simulative and non-simulative categories. The non-simulative method in [16] estimates the power consumption from an information theoretical point of view. In [14], the authors describe a strategy called Dual Bit Type (DBT) model where not only the random activity of the least significant bits, but also the correlated activity of the most significant bits is taken into account. The method in [10] proposes a modeling approach for functional units that are typically used in digital signal processing systems, such as adders, multipliers and delay elements. Thereby, a 4-dimensional table-based [9] macro model is used by the authors.

Also, some works [3] focused on transformations at the algorithmic and the architectural level to obtain low power designs. In [2], transformations for nested loop programs are discussed. In [4, 18, 19, 21], several scheduling and binding techniques for low power are studied. Some energy estimations for processor arrays with hierarchical memory structures are made in [6].

However, to the best of our knowledge, our work presented here is the first which considers the relationship between space-time mappings of computation intensive algorithms and energy consumption. Here, we specify a power-consumption model used in the methodology described afterwards for energy estimation of piecewise regular processor arrays.

3 Notation and Background

3.1 Algorithms

The class of algorithms we are dealing with in this paper is a class of recurrence equations defined as follows:

Definition 1. (*Piecewise Regular Algorithm*). *A piecewise regular algorithm contains N quantified equations*

$$S_1 [I], \dots, S_i [I], \dots, S_N [I]$$

Each equation $S_i [I]$ is of the form

$$x_i [I] = f_i (\dots, x_j [I - d_{ji}], \dots)$$

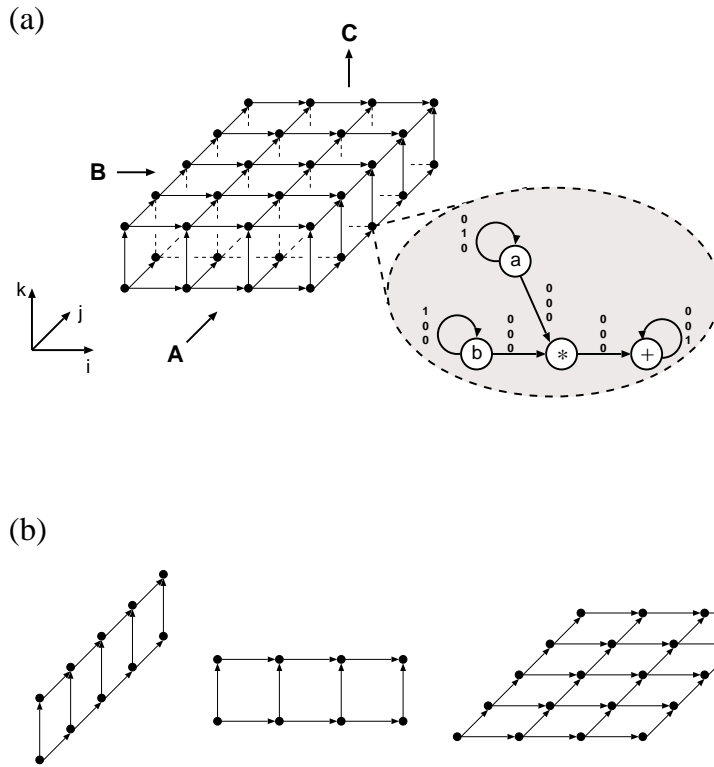


Fig. 1. In (a), an index space and the reduced dependence graph is shown. Some possible mappings are depicted in (b).

where $I \in \mathcal{I}_i \subseteq \mathbb{Z}^n$, $x_i[I]$ are indexed variables, f_i are arbitrary functions, $d_{j_i} \in \mathbb{Z}^n$ are constant data dependence vectors, and \dots denote similar arguments.

The domains \mathcal{I}_i are called index spaces, and in our case defined as follows:

Definition 2. (*Linearly Bounded Lattice*). A linearly bounded lattice denotes an index space of the form

$$\mathcal{I} = \{I \in \mathbb{Z}^n \mid I = M\kappa + c \wedge A\kappa \geq b\}$$

where $\kappa \in \mathbb{Z}^l$, $M \in \mathbb{Z}^{n \times l}$, $c \in \mathbb{Z}^n$, $A \in \mathbb{Z}^{m \times l}$ and $b \in \mathbb{Z}^m$. $\{\kappa \in \mathbb{Z}^l \mid A\kappa \geq b\}$ defines an integral convex polyhedron or in case of boundedness a polytope in \mathbb{Z}^l . This set is affinely mapped onto iteration vectors I using an affine transformation ($I = M\kappa + c$).

Throughout the paper, we assume that the matrix M is square and invertible. Then, each vector κ is uniquely mapped to an index point I . Furthermore, we require that the index space is bounded.

For illustration purposes throughout the paper, the following example is used.

Example 1. The well known matrix multiplication algorithm computes the product $C = A \cdot B$ of two matrices $A \in \mathbb{R}^{N_1 \times N_3}$ and $B \in \mathbb{R}^{N_3 \times N_2}$ and is defined as follows

$$c_{ij} = \sum_{k=1}^{N_3} a_{ik} b_{kj} \quad \forall 1 \leq i \leq N_1 \wedge 1 \leq j \leq N_2.$$

A corresponding piecewise regular algorithm is given by

input operations

$$\begin{aligned} a[i, 0, k] &\leftarrow a_{ik} & 1 \leq i \leq N_1 \wedge 1 \leq k \leq N_3 \\ b[0, j, k] &\leftarrow b_{kj} & 1 \leq j \leq N_2 \wedge 1 \leq k \leq N_3 \\ c[i, j, 0] &\leftarrow 0 & 1 \leq i \leq N_1 \wedge 1 \leq j \leq N_2 \end{aligned}$$

computations

$$\begin{aligned} a[i, j, k] &\leftarrow a[i, j-1, k] & \forall (i \ j \ k)^T = I \in \mathcal{I} \\ b[i, j, k] &\leftarrow b[i-1, j, k] & \forall (i \ j \ k)^T = I \in \mathcal{I} \\ z[i, j, k] &\leftarrow a[i, j, k] \cdot b[i, j, k] & \forall (i \ j \ k)^T = I \in \mathcal{I} \\ c[i, j, k] &\leftarrow c[i, j, k-1] + z[i, j, k] & \forall (i \ j \ k)^T = I \in \mathcal{I} \end{aligned}$$

output operations

$$c_{ij} \leftarrow c[i, j, N_3] \quad 1 \leq i \leq N_1 \wedge 1 \leq j \leq N_2$$

The data dependence vectors are $d_{aa} = (0 \ 1 \ 0)^T$, $d_{bb} = (1 \ 0 \ 0)^T$, $d_{cc} = (0 \ 0 \ 1)^T$, $d_{az} = (0 \ 0 \ 0)^T$, $d_{bz} = (0 \ 0 \ 0)^T$, and $d_{zc} = (0 \ 0 \ 0)^T$. The index space is given by

$$\mathcal{I} = \{I = (i \ j \ k)^T \in \mathbb{Z}^3 \mid 1 \leq i \leq N_1 \wedge 1 \leq j \leq N_2 \wedge 1 \leq k \leq N_3\}.$$

Computations of piecewise regular algorithms may be represented by a *dependence graph* (DG). The dependence graph of the algorithm of Example 1 is shown in Fig. 1 (a). The dependence graph expresses the partial order between the operations. Each variable of the algorithm is represented at every index point $I \in \mathcal{I}$ by one node. The edges correspond to the data dependencies of the algorithm. They are *regular* throughout the algorithm, i.e., $a[i, j, k]$ is directly dependent on $a[i, j-1, k]$. The dependence graph specifies implicitly all legal execution orderings of operations: if there is a directed path in the dependence graph from one node $a[J]$ to a node $z[K]$ where $J, K \in \mathcal{I}$, then the computation of $a[J]$ must precede the computation of $z[K]$.

Henceforth, and without loss of generality¹, we assume that all indexed variables are embedded in a common index space \mathcal{I} . Then, the corresponding dependence graphs can be represented in a reduced form.

Definition 3. (*Reduced Dependence Graph*). A reduced dependence graph (RDG) $G = (V, E, D)$ of dimension n is a network where V is a set of nodes and $E \subseteq V \times V$ is a set of edges. To each edge $e = (v_i, v_j)$ there is associated a dependence vector $d_{ij} \in D \subset \mathbb{Z}^n$.

The RDG of the matrix multiplication algorithm is shown in Fig. 1 (a). Each node v in the graph corresponds to one equation in the section computations of the algorithm.

¹ All described methods can also be applied for each quantification individually.

3.2 Space-Time Mapping

Linear transformations as in Eq. (1), are used as *space-time mappings* [12,17] in order to assign a *processor index* $p \in \mathbb{Z}^{n-1}$ (space) and a *sequencing index* $t \in \mathbb{Z}$ (time) to index vectors $I \in \mathcal{I}$.

$$\begin{pmatrix} p \\ t \end{pmatrix} = TI = \begin{pmatrix} Q \\ \lambda \end{pmatrix} I \quad (1)$$

In Eq. (1), $Q \in \mathbb{Z}^{(n-1) \times n}$ and $\lambda \in \mathbb{Z}^{1 \times n}$. The main reasons for using linear allocation and scheduling functions is that the data flow between PEs is local and regular which is essential for low power VLSI implementations. The interpretation of such a linear transformation is as follows: The set of operations defined at index points $\lambda \cdot I = \text{const.}$ are scheduled at the same time step. The index space of allocated processing elements (*processor space*) is denoted by \mathcal{Q} and is given by the set $\mathcal{Q} = \{p \mid p = Q \cdot I \wedge I \in \mathcal{I}\}$. This set can also be obtained by choosing a projection of the dependence graph along a vector $u \in \mathbb{Z}^n$, i.e., any coprime² vector u satisfying $Q \cdot u = 0$ [12] describes the allocation equivalently.

Allocation and scheduling must satisfy that no data dependencies in the DG are violated. This is ensured by the following *causality constraint*

$$\lambda \cdot d_{ij} \geq 0 \quad \forall (v_i, v_j) \in E. \quad (2)$$

A sufficient condition for guaranteeing that no two or more index points are assigned to a processing element at the same time step is given by

$$\text{rank} \begin{pmatrix} Q \\ \lambda \end{pmatrix} = n. \quad (3)$$

Using the projection vector u satisfying $Q \cdot u = 0$, this condition is equivalent to $\lambda \cdot u \neq 0$ [24].

Definition 4. (*Iteration Interval*) [26]. *The iteration interval π of an allocated and scheduled piecewise regular algorithm is the number of time instances between the evaluation of two successive instances of a variable within one processing element.*

Definition 5. (*Block Pipelining Period*) [13]. *The block pipelining period of an allocated and scheduled piecewise regular algorithm is the time interval between the initiations of two successive problem instances and is denoted by β .*

Lets consider the matrix multiplication algorithm introduced in Example 1 as a problem instance. The whole matrices A and B have to read into the processor array before the next pair can be read, the time between these input operations is the block pipelining period β . Let λ be the schedule vector. Then, the block pipelining period β may be computed as follows,

$$\beta = \max_{I_1 \in \mathcal{I}} \{\lambda \cdot I_1\} - \min_{I_2 \in \mathcal{I}} \{\lambda \cdot I_2\} = \max_{I_1, I_2 \in \mathcal{I}} \{\lambda(I_1 - I_2)\}.$$

² A vector x is said to be *coprime* if the absolute value of the greatest value of the greatest common divisor of its elements is one.

4 Power Modeling and Energy Estimation

In digital CMOS circuits, the dominant source of power consumption is switching power [22]. The average power consumed by a CMOS gate can be computed using the following equation,

$$P_{sw} = \frac{1}{2}C_L V_{dd}^2 N f,$$

where C_L is the gate output load capacitance, V_{dd} is the supply voltage, f is the clock frequency, and N is the average or expected number of output transitions per clock cycle.

Due to the influence of the switching activity on the power consumption, our main idea is to exploit the fact that power consumption is drastically reduced when some inputs of a functional unit remain unchanged for $n > 1$ clock cycles.

Here, we want to discuss the impact of the space-time mapping on the power and energy consumption respectively of the resulting processor array. Our approach identifies regions with decreased switching activity of functional units' input operands and take these power savings into account. An estimation methodology is presented in the following. This methodology estimates for a given piecewise regular algorithm and a space-time mapping T the average power consumption of the entire array.

Briefly described this methodology can be subdivided into two hierarchical estimation steps,

- PE-level power estimation,
- array-level power estimation.

4.1 PE-level Power Estimation

A sketch of a typical processor element's internal structure is shown in Fig. 2. It consists of a core part where all the functional units are located, a controller, and some delay registers. In the final paper version, we quantify the percentages

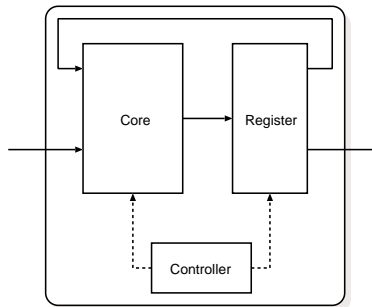


Fig. 2. Schematically internal structure of one processor element.

of power consumption for the functional units P_{FU} , the control structures P_{Ctrl} , and the registers P_{Rg} and these parts' proportion of the overall power consumption of one processing element. Then the power consumption of one PE can be approximated as follows, $P_{FU}(\lambda, u) = P_{FU}(u) + P_{Ctrl}(\lambda) + P_{Rg}(\lambda)$.

For characterization of the functional units (adders, multipliers, etc.), standard register-transfer level power estimation tools from Synopsys [23] are used.

Table 1. Average power consumption of different functional units.

n	$P_{avg,A}$	$P_{avg,B}$	$P_{avg,C}$	$P_{avg,D}$
1	26.97 μ W	204.2 μ W	212.0 μ W	319.6 μ W
2	22.33 μ W	155.4 μ W	164.0 μ W	225.0 μ W
3	18.82 μ W	138.6 μ W	145.6 μ W	190.1 μ W
4	16.99 μ W	129.6 μ W	137.3 μ W	175.1 μ W
5	16.31 μ W	125.4 μ W	133.8 μ W	164.3 μ W
6	15.68 μ W	120.5 μ W	128.4 μ W	159.4 μ W
7	15.48 μ W	119.5 μ W	125.2 μ W	153.3 μ W
8	15.29 μ W	116.8 μ W	124.4 μ W	151.6 μ W
9	15.09 μ W	116.3 μ W	123.7 μ W	147.8 μ W
10	14.89 μ W	115.5 μ W	122.7 μ W	145.8 μ W
∞_0	8.49 μ W	-	-	-

In Table 1, the average power consumption of some 16-bit functional units are listed (A = ripple-carry adder, B = carry-save array multiplier, C = carry-save array multiplier with two pipeline stages, D = Wallace-tree multiplier with three pipeline stages). Each functional unit has two input operands. The value of one operand is assumed to be constant for n clock cycles; the other can change randomly in every clock cycle. These values are visualized in Fig. 3 (a) for the 16-bit ripple-carry adder and Fig. 3 (b) for the multipliers respectively. The curves are derived by regression, where the function is of type $P = a_0 + a_1e^{-n} + a_2ne^{-n} + a_3n^2e^{-n}$. The regression is good enough to have errors less than 2%. Since we are only interested in integer multiples of the clock cycle for n , the derived models may be stored in a table without too much effort.

4.2 Array-level Power Estimation

Based on the class of piecewise regular algorithms, we want to estimate the power consumption for a given space-time mapping $T = (Q \lambda)^T$. It is obvious that the cost (number of processor elements) and the latency is influenced by the space-time mapping. In earlier work [11], we described how to determine the cost and the latency as a measure for performance. Here, we just briefly outline the main ideas. If we assume that processor arrays are resource-dominant, we are able to approximate the cost as being proportional to the processor count. *Ehrhart polynomials* [5, 7] may be evaluated to count the number of points (processor elements, $\#PE$) in the projected index space.

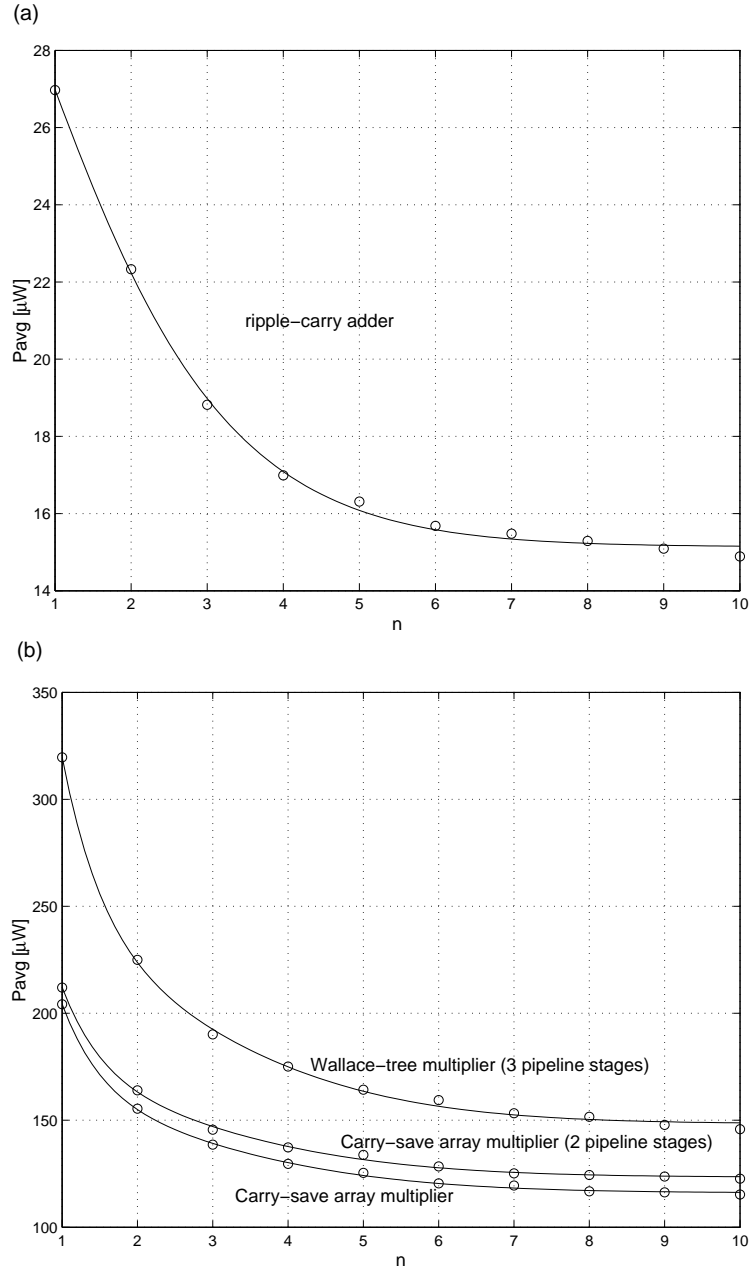


Fig. 3. Average power consumption of some 16-bit functional units when one operand is constant for n clock cycles and the other can change randomly in every clock cycle.

The latency is determined by solving a minimization problem which may be formulated as a mixed integer linear program (MILP) [25, 26]. Also, modified low power scheduling and binding techniques like in [19, 21] can be applied to compute a suited schedule.

Here, we want to discuss the impact of the space-time mapping on the power and energy consumption respectively of the resulting processor array. Our approach identifies regions with decreased switching activity of functional units' input operands and take these power savings into account. An estimation algorithm is presented on the following pages. The algorithm estimates for a given RDG G , an index space \mathcal{I} , a space-time mapping T , the number of processor elements $\#PE$, and the block pipelining period β the average power consumption P_{array} of the entire array. The processor count $\#PE$ and the block pipelining period β of the array may be computed as described earlier in this paper.

Once, the average power consumption P_{array} of the entire processor array is estimated, the energy consumption per problem instance is computed as follows,

$$E = \beta \cdot P_{array}.$$

Without loss of generality, we assume in the following that the iteration period π is one and that each RDG node is mapped onto a dedicated resource. Our estimation algorithm can be subdivided into two phases. In the first phase, the worst case power consumption is computed, i.e., when the switching activity of all functional units' input operands is highest. Therefore, the power consumption P_{PE} of one processor element is determined by summation of the power consumption $P_{v_i}(1)$ of all of its FUs

$$P_{PE} = \sum_{\forall v_i \in V} P_{v_i}(1).$$

The one in the term $P_{v_i}(1)$ denotes that operands can change in every clock cycle.

POWER ESTIMATION

```

1  IN:   RDG  $G$ ,  $\mathcal{I}$ ,  $T = \begin{pmatrix} Q \\ \lambda \end{pmatrix}$ ,  $\#PE$ , and  $\beta$ 
2  OUT:  $P_{array}$ 
3  BEGIN
4      $P_{PE} \leftarrow 0$ 
5     FOR all nodes  $v \in G$  DO
6          $P_{v,1} \leftarrow \text{lookUpPower}(v, 1)$ 
7          $P_{PE} \leftarrow P_{PE} + P_{v,1}$ 
8     ENDFOR
9      $P_{array} \leftarrow \#PE \cdot P_{PE}$ 
10    FOR all edges  $e \in G$  DO
11         $d$  is dependence vector of edge  $e$ 
12        node  $v \leftarrow \text{source}(e)$ 
13        node  $w \leftarrow \text{target}(e)$ 

```

10

```

14 IF ( $v = w$ ) THEN
15   IF ( $S_v$  is propagation equation) THEN
16     IF ( $Q \cdot d = \mathbf{0}$ ) THEN
17       FOR all adjacent edges  $e'$  of  $v$ 
18          $d'$  is dependence vector of edge  $e'$ 
19         IF ( $d' = \mathbf{0}$ ) THEN
20            $w \leftarrow \text{target}(e')$ 
21            $P_{w,1} \leftarrow \text{lookUpPower}(w, 1)$ 
22            $P_{w,\beta} \leftarrow \text{lookUpPower}(w, \beta)$ 
23            $P_{\text{array}} \leftarrow P_{\text{array}} - \#PE \cdot (P_{w,1} - P_{w,\beta})$ 
24         ENDIF
25       ENDFOR
26     ENDIF
27   ELSE
28      $(k, m) \leftarrow \text{getOperandFixedCycles}(T, v)$ 
29      $P_{w,1} \leftarrow \text{lookUpPower}(w, 1)$ 
30      $P_{w,k} \leftarrow \text{lookUpPower}(w, k)$ 
31      $P_{\text{array}} \leftarrow P_{\text{array}} - m \cdot (P_{w,1} - P_{w,k})$ 
32   ENDIF
33 ENDIF
34 ENDFOR
35 END

```

Subsequently, the power consumption of the entire array is obtained by extrapolation of this value. In the second algorithm phase array regions with lower switching activity are detected. Therefore, the whole reduced dependence graph is traversed to examine self-loops³. These self-loops correspond to inputs of a processor element. If these inputs remain unchanged for more than one period, the switching activity is decreased and consequently also the power. It remains to determine for how long inputs are constant and how many processor elements are affected. Two cases can be differentiated:

1. **Propagation equations mapped onto itself.** Propagation equations are only used to distribute data from one processor to another. Due to the regularity and locality of the considered processor arrays, they occur very commonly. If such a propagation equation is mapped onto itself ($Q \cdot d = \mathbf{0}$) no data transport is needed, i.e., the data remains in one processor element unchanged for β cycles until the next problem instance is fed into the array. Thus, the switching activity of all adjacent nodes v_i (functional units) in the same processor element is reduced. Therefore, the estimation of the average power consumption is decreased by $P_{v_i}(1) - P_{v_i}(\beta)$. As a propagation equation has global influence the activity is reduced in every processor element ($\#PE$).
2. **Other self-loops.** These are the remaining inputs which may be constant for k clock cycles. The number of processor elements with these constant

³ A self-loop is an edge where source and target node are the same.

inputs is denoted by m . Let \mathcal{I}_{in_1} be the input index space of variable in_1 . Transforming this index space by Q and counting the number of points in the transformed space, gives m .

$$m = |\{I \in \mathbb{Z}^{n-1} | I = Q \cdot I_{in_1} \wedge I_{in_1} \in \mathcal{I}_{in_1}\}|$$

This counting problem is similar to the earlier described one and can also be solved by using Ehrhart polynomials. Once k and m are determined, the overall estimated power consumption can be reduced by $m \cdot (P_{in_1}(1) - P_{in_1}(k))$.

In the next section the overall algorithm is explained by means of discussing some results.

5 Results

Reconsider the introductory Example 1. As an allocation we choose for the addition a 16-bit ripple-carry adder and for the multiplication a three-stage pipelined Wallace-tree multiplier. The input operations a and b are mapped each to one resource of type *input*. The execution times of these operations are zero. This is equivalent to a multi-cast without delay to a set of processors. Furthermore, let $u = (1 \ 0 \ 0)^T$ be the chosen projection vector. Then, after scheduling and cost calculus, we obtain the schedule vector $\lambda = (1 \ 0 \ 1)$ and as cost $\#PE = N_2 \cdot N_3$. Now, with this information we are able to estimate the power consumption by applying the proposed algorithm. First, the worst case power consumption is determined, i.e., the switching activity of functional units' when input operands change each both each cycle. Second, in the main part of the algorithm, two types of equations with lower input activity are detected and the overall power consumption is adapted.

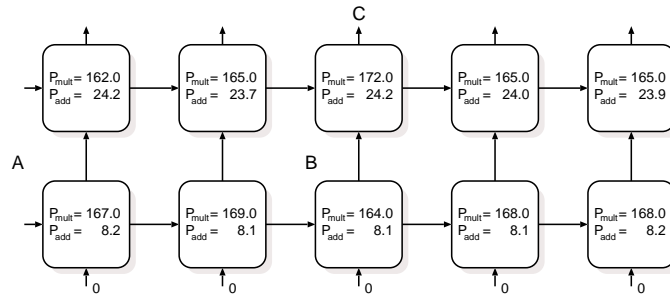


Fig. 4. Processor array for $u = (1 \ 0 \ 0)^T$, $N_1 = 4$, $N_2 = 5$, and $N_3 = 2$.

The processor array for a projection in direction $u = (1 \ 0 \ 0)^T$ is shown in Fig. 4. Due to this projection, the variable b is mapped onto itself. From this it

follows that one operand of the multiplication remains unchanged for some time. At the beginning of a computation, the whole matrix B is input simultaneously to the array, whereas the matrix A is fed sequentially row by row from the left side into the array. Since the matrix A has N_1 rows, one operand of the multiplier is fixed for $\beta = N_1$ clock cycles which significantly reduces the power consumption in the multipliers by 45% (see Table 1). On account of the design regularity the power savings can be multiplied by $\#PE$ (line 23 of the algorithm). The second point where less power is consumed is the constant input variable c . One input of the adders in the lower row of the processor array is permanently zero. This partial area with reduced power consumption in the array is determined by the function `getOperandFixedCycles`. In addition to the time ($k = \infty$) where one input remains unchanged, the number $m = N_2$ of processors with reduced switching activity is returned.

Table 2. Average power and energy consumption of different mappings.

u	P_{sim} [μW]	P_{ext} [μW]	Err_{ext} [%]	P_{est} [μW]	Err_{est} [%]	E_{sim} [pJ]	E_{est} [pJ]
$(1\ 0\ 0)^T$	2020	3466	71.6	1928	-4.6	80.8	77.1
$(0\ 1\ 0)^T$	1530	2773	81.2	1456	-4.8	76.5	72.8
$(0\ 0\ 1)^T$	7260	6931	-4.5	6931	-4.5	145.2	138.6

In Table 2, the power consumption for different projection vectors is shown, where for illustration purposes, the upper boundaries of the index space are set to $N_1 = 4$, $N_2 = 5$, and $N_3 = 2$. In the table, P_{sim} is the exact value obtained by simulation of the entire array. The worst case extrapolation (line 4–9 in the algorithm) is denoted by P_{ext} . The power consumption of our estimation algorithm is labeled with P_{est} . Whereas the simple extrapolation method has errors up to 81%, our approach is very accurate with errors less than 5%.

Furthermore, the energy values per matrix multiplication in the table show the significant influence of the chosen space-timing mapping. Different mappings can lead to energy consumptions which differ up to a factor of two.

6 Conclusions and Future Work

A first study of a matrix multiplication algorithm has shown the great impact of a chosen mapping to the average energy consumption of the resulting array and the accuracy (errors $< 5\%$) of our estimation approach when comparing it with RTL power estimation tools from Synopsys [23].

Furthermore, our methodology is independent of the problem (array) size, since, an estimation with Synopsys design tools has linear time and memory complexity in dependence on the number of processor elements. Power estimation for large processor arrays using the Synopsys design tools rapidly becomes crucial

since memory usage is growing to GBytes and estimation time to several hours. Exact comparisons of the complexity and also a quantification of the percentages of power consumption for the functional units, the controller, and the registers and these parts' proportion of the overall power consumption of one processing element are presented in the final version of this paper. First experiments of matrix multiplication and LU decomposition have shown that since all data is stored locally inside processor element's registers, the part of the register power consumption averages from $\sim 10 - 15\%$ of the overall power consumption.

Finally (in the final paper), our methodology will be verified for a piecewise regular algorithm in a case study for LU decomposition. In Fig. 5, a piecewise

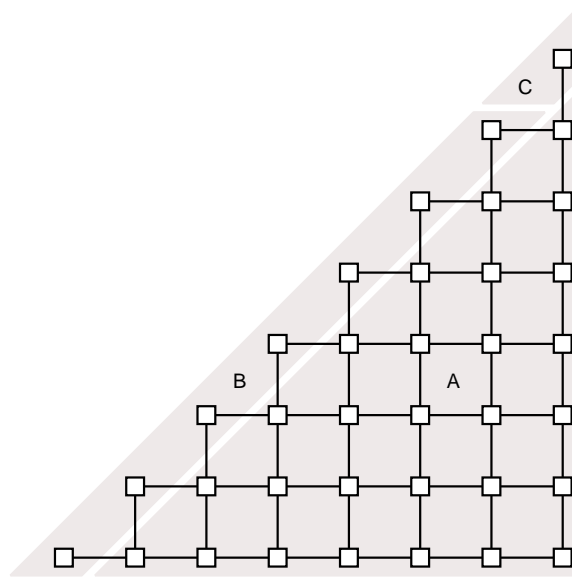


Fig. 5. Sketch of piecewise regular processor array for LU decomposition.

regular processor array for LU decomposition is schematically shown. This array can be subdivided in three pieces, where the parts *A* and *B* also change their functionality over the time.

Our new estimation methodology is currently integrated into the PARO design system and can be used during the process of automated synthesis of regular circuits. PARO is a design system project for modeling, transforming, optimization, and processor synthesis for the class of piecewise linear algorithms [1, 20].

References

1. Marcus Bednara and Jürgen Teich. Synthesis of FPGA Implementations from Loop Algorithms. In *First International Conference on Engineering of Reconfigurable*

- Systems and Algorithms (ERSA'01)*, pages 1–7, Las Vegas, NV, June 2001.
2. Francky Catthoor, Frank Franssen, Sven Wuytack, Lode Nachtergaele, and Hugo De Man. Global Communication and Memory Optimizing Transformations for Low Power Systems. In *VLSI Signal Processing Workshop*, pages 178–187, October 1994.
 3. Anantha P. Chandrakasan, Miodrag Potkonjak, Renu Mehra, Jan Rabaey, and Robert W. Brodersen. Optimizing Power Using Transformations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(1):12–31, January 1995.
 4. Jui-Ming Chang and Massoud Pedram. Module Assignment for Low Power. In *IEEE European Design Automation Conference (EuroDAC)*, pages 376–381, Geneva, Switzerland, September 1996.
 5. Philippe Clauss and Vincent Loechner. Parametric Analysis of Polyhedral Iteration Spaces. *Journal of VLSI Signal Processing*, 19(2):179–194, July 1998.
 6. Uwe Eckhardt. *Algorithmus-Architektur-Codesign für den Entwurf digitaler Systeme mit eingebettetem Prozessorarray und Speicherhierarchie*. PhD thesis, Technische Universität Dresden, Fakultät Elektrotechnik, Dresden, Germany, 2000.
 7. Eugène Ehrhart. *Polynômes arithmétiques et Méthode des Polyèdres en Combinatoire*, volume 35 of *International Series of Numerical Mathematics*. Birkhäuser Verlag, Basel, 1. edition, 1977.
 8. Paul Feautrier. Automatic Parallelization in the Polytope Model. Technical Report 8, Laboratoire PRiSM, Université des Versailles St-Quentin en Yvelines, 45, avenue des États-Unis, F-78035 Versailles Cedex, June 1996.
 9. Subodh Gupta and Farid N. Najm. Power Macro-Models for DSP Blocks with Application to High-Level Synthesis. In *IEEE International Symposium on Low Power Electronics and Design*, pages 103–105, San Diego, CA, August 1999.
 10. Subodh Gupta and Farid N. Najm. Power Modeling for High-Level Power Estimation. *IEEE Transactions on Very Large Integration (VLSI) Systems*, 8(1):18–29, February 2000.
 11. Frank Hannig and Jürgen Teich. Design Space Exploration for Massively Parallel Processor Arrays. In Victor Malyskin, editor, *Parallel Computing Technologies, 6th International Conference, PaCT 2001, Proceedings*, volume 2127 of *Lecture Notes in Computer Science (LNCS)*, pages 51–65, Novosibirsk, Russia, September 2001. Springer.
 12. Robert H. Kuhn. Transforming Algorithms for Single-Stage and VLSI Architectures. In *Workshop on Interconnection Networks for Parallel and Distributed Processing*, pages 11–19, West Lafayette, IN, April 1980.
 13. Sun-Yuan Kung. *VLSI Array Processors*. Prentice Hall, Englewood Cliffs, New Jersey, 1987.
 14. Paul E. Landman and Jan M. Rabaey. Architectural Power Analysis: The Dual Bit Type Method. *IEEE Transactions on Very Large Integration (VLSI) Systems*, 3(2):173–187, June 1995.
 15. Christian Lengauer. Loop Parallelization in the Polytope Model. In Eike Best, editor, *CONCUR'93, Lecture Notes in Computer Science 715*, pages 398–416. Springer-Verlag, 1993.
 16. Diana Marculescu, Radu Marculescu, and Massoud Pedram. Information Theoretic Measures for Power Analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(6):599–610, June 1996.
 17. Dan I. Moldovan. On the Design of Algorithms for VLSI Systolic Arrays. In *Proceedings of the IEEE*, volume 71, pages 113–120, January 1983.

18. Enric Musoll and Jordi Cortadella. High-level Synthesis Techniques for Reducing the Activity of Functional Units. In *International Symposium on Low-Power Design*, pages 99–104, April 1995.
19. Enric Musoll and Jordi Cortadella. Scheduling and Resource Binding for Low Power. In *Int. Symp. on System Synthesis*, pages 104–109, 1995.
20. PARO Design System Project. <http://www-date.upb.de/research/paro/>.
21. Anand Raghunathan and Niraj K. Jha. An ILP Formulation for Low Power based on Minimizing Switched Capacitance during Data Path Allocation. In *IEEE Symposium on Circuits and Systems*, May 1995.
22. Anand Raghunathan, Niraj K. Jha, and Sujit Dey. *High-Level Power Analysis and Optimization*. Kluwer Academic, Norwell, Massachusetts, 1998.
23. Synopsys Inc. <http://www.synopsys.com>.
24. Jürgen Teich. *A Compiler for Application-Specific Processor Arrays*. PhD thesis, Institut für Mikroelektronik, Universität des Saarlandes, Saarbrücken, Germany, 1993.
25. Jürgen Teich, Lothar Thiele, and Li Zhang. Scheduling of Partitioned Regular Algorithms on Processor Arrays with Constrained Resources. *Journal of VLSI Signal Processing*, 17(1):5–20, September 1997.
26. Lothar Thiele. Resource Constrained Scheduling of Uniform Algorithms. *Journal of VLSI Signal Processing*, 10:295–310, 1995.