

Energy/Power Estimation of Regular Processor Arrays

Steven Derrien
IRISA, France,
sderrien@irisa.fr

Sanjay Rajopadhye
Colorado State University
svr@cs.colostate.edu

ABSTRACT

We propose a high-level analytical model for estimating the energy and/or power dissipation in VLSI processor (systolic) array implementations of loop programs, particularly for implementations on FPGA based CO-processors. We focus on the respective impact of the array design parameters on the overall off-chip I/O traffic and the number and sizes of the local memories in the array. The model is validated experimentally and shows good results (12.7% RMS error in the predictions).

Keywords

Power Estimation, Programmable logic, Design Space Exploration, Processor Array Partitioning

1. INTRODUCTION

Designing high performance embedded systems is a hard challenge, especially in the face of fact that the designer has to take into consideration, a number of *performance* constraints. We view performance as some weighted combination of three key criteria—speed, cost (i.e., area) and power/energy. Among these design challenges, the ability to estimate and tune the power consumption at a high level of the design flow is a clear advantage.

In most embedded systems, there are a number of compute intensive kernel programs, and it is essential to obtain the maximal performance for these kernels. The problem of mapping such compute intensive kernels directly to hardware in the form of VLSI processor arrays has received considerable attention for over two decades, starting from the days of systolic arrays and synthesis. While many questions still remain unresolved, the field has attained relative maturity, and Industrial tools such as PiCo [6] are now available. An important part of these tools is a design space exploration engine, which searches for implementations of a given program that optimize performance criteria.

In this paper, we propose an analytical model for the power/energy consumption associated with such VLSI processor

arrays. Our goal is to investigate the impact of partitioning transformations (and their associated parameters) on the design power consumption. Such a model could then be used to guide the design space exploration in order to determine the best design, given power, area and performance constraints. As of now, our model and its experimental validation focuses on FPGAs based system. However, we believe that it is generic enough to be applied to full-custom implementations with only slight modification.

This model is based on the following observation : in current data dominated embedded applications [3], the two main contributions to power are associated to (i) off-chip I/O transfers, and (ii) on-chip memory modules. Since the processor array partitioning parameters have a strong influence over both global off-chip I/O volume and on-chip processor local memory size, we expect that a trade-off exists between the two, and confirm this hypothesis with some experimental validation.

The paper is organized as follows. Section 2 describes the basis of VLSI processor array synthesis, loop parallelization and array partitioning. Section 3 presents our energy dissipation analytical model which merges on-chip and off-chip energy dissipation models, and section 4 provides some preliminary experimental results. Conclusion and future work are given in section 5.

2. BACKGROUND

Synthesizing regular VLSI processor arrays from nested loop programs consists of three steps: loop parallelization to obtain a “virtual” processor array, followed by “partitioning” and “clustering” to satisfy resource constraints.

2.1 Loop parallelization

We consider perfectly nested loop nests with uniform (or easily “uniformizable”) dependences, and polyhedral loop bounds with size known at compile time.

Let I denote the loop index domain which is defined by a set of linear of affine inequalities and \vec{x} denote the loop index vector $[x_1 \dots x_n]$, we have $\vec{x} \in I$. Each array variable accessed in the loop body can be characterized by m data dependence vectors represented as a matrix $D = [\vec{d}_1 \dots \vec{d}_m]$.

Consider a 256×256 matrix multiplication kernel as given below. The loop domain is a $3D$ cube, and three distinct arrays are accessed within the loop body: input matrixes A and B and output matrix C. One data dependence exists for each variable $\vec{d}_A = [1 \ 0 \ 0]$, $\vec{d}_B = [0 \ 1 \ 0]$, and $\vec{d}_C = [0 \ 0 \ 1]$.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ISSS'02, October 2–4, 2002, Kyoto, Japan.

Copyright 2002 ACM 1-58113-576-9/02/0010 ...\$5.00.

```

for (i=0;i<256;i++) for (j=0;j<256;j++) {
  c[i][j]=0;
  for (k=0;k<256;k++)
    C[i][j] = C[i][j] + A[i][k]*B[k][j];
}

```

Following well established methods, the parallelization of such a loop nest consists of two steps: (i) the *scheduling* function maps iteration \vec{x} to time instant $t = \tau(\vec{x})$ and (ii) the *allocation* assign iteration \vec{x} to a PE $\vec{p} = \pi(\vec{x})$ in the processor space. We assume these two functions to be linear, and we can hence combine them into a single transformation using a matrix Π , defined by $\Pi = \begin{bmatrix} \tau \\ \pi \end{bmatrix}$. We can now express the whole transformation as $\vec{x}' = \Pi\vec{x}$.

The transformation matrix Π must obey several constraints apply to this matrix: (i) the schedule must be valid with respect to data dependencies which translates as $\vec{\tau}d_k > 0$ for each dependence vector d_k (ii) only one iteration can be allocated to PE \vec{p} at time t , which translates as $\det \Pi \neq 0$.

We impose two additional restrictions over π and τ : (i) we only allow unidirectional communications in our array and (ii) we require that the delay of an interconnection link is at least as large as the Manhattan distance between the PE's involved.

For our matrix multiplication example, let us choose $\pi = [0 \ 0 \ 1]$ and $\tau = [1 \ 1 \ 1]$. We can now determine (i) the array topology and (ii) the internal PE architecture. We obtain a $N \times N$ rectangular array of PE's each executing a simple 16 bit MAC operation every cycle, with 2×16 and with 1×32 bits registers (for A, B and C). We will make the distinction between two types of registers: (i) *Temporal registers* which act as local memory within the PE data-path, and *Spatial registers* which connect two PE's along one processor space dimension.

The array derived by the above process is generally impractical, since the number of processors depends on the problem size (eg. 64k processors for our matrix multiplication example). In order to obtain practical arrays (i.e., those that respect silicon resource constraints, we use two additional transformations, in combination with each other.

2.2 Tiling (LPGS)

Tiling consists of “cutting” the space-time domain using a set of hyper-planes and thus forming parallelepipedic sub-domains (or tiles). Tiles are considered as atomic execution units and are executed sequentially on the target architecture. In the context of regular array synthesis, tiling partitions the n -dimensional space-time domain using using $n - 1$ hyper-planes whose normal vectors are a linear combination of the processor space base vector, so that each tile can be executed sequentially on a fixed size processor array (*LPGS partitioning*).

We restrict normal vectors to the tiling the hyper-planes to be scaled basis vectors of the processor space. This will only allow rectangular tile shapes with communication channels normal to the tile boundaries. This transformation can then be represented by a diagonal matrix Ω whose coefficients ω_i correspond to the tiling scale factor along processor axis i .

To ensure atomicity, the tiling must satisfy certain constraints. Specifically, Ω must satisfy $\Omega \cdot (\pi D) \geq 0$, i.e., LPGS partitioning is only valid for array with unidirectional data flow. Our previous constraints on Π ensure that this is al-

ways satisfied.

Once we have chosen the tile shape, the computations are executed using a succession of passes, each of them executed on a processor tile (or sub-array), the number of PE's within each sub-array being given by $n_p = \det |\Omega|$. Again, such a sequential execution must obey some scheduling constraints, so that tile level data dependencies are satisfied. Let denote τ_Ω the tile level scheduling vector, τ_Ω must then be chosen such that $\tau_\Omega \cdot D \succeq 0$.

Although this transformation has the advantage of allowing (without any change to the original PE architecture) the emulation of an arbitrary sized processor array, tile intermediate results must be stored in an external memory so that they can be used when needed by a following tile. This results in an increase in the overall I/O traffic.

To have a better understanding of this transformation let us apply the tiling transformation to our running example. As seen in 2.1, we obtain a 2D PE array from the mapping step. Hence two single hyperplanes are used to perform the tiling transformation, and the matrix Ω associated with the tiling proposed in figure1 would have as diagonal components $\omega_1 = 2$ and $\omega_2 = 2$. These values of ω_i determine the number of processors to be implemented for each pass $n_p = \det |\Omega| = \omega_1 \omega_2$. The total number of passes required to perform the whole loop nest is then $P = \lceil \frac{256}{\omega_1} \rceil \lceil \frac{256}{\omega_2} \rceil$.

2.3 Clustering (LSGP)

Clustering is another well known space-time transformation [5, 7]. The key idea is to group PE's derived from the systolisation process into “clusters” in which the original PE's iterations are performed sequentially. Clustering actually consists of two coupled sub-problems (i) defining the cluster geometry (ii) specifying the schedule within clusters.

To determine the cluster shape, as it is the case for tiling, we partition the processor space domain using a set of $n - 1$ hyperplanes which are chosen such that their normal vector is a combination of the processor base vector. We will again impose that the normal vectors of these hyperplanes are the processor space base vector.

The cluster transformation can hence be represented as an $n - 1 \times n - 1$ diagonal matrix Γ with positive integer elements σ_i . The number of PE emulated by a cluster is then given by $\det |\Gamma| = \prod_{i=1}^{n-1} \sigma_i$.

Once the cluster geometry is set, iterations within this cluster have to be scheduled. In this work, we will stick to very simple scheduling: iterations within a cluster are executed in an axis-major order as if we unrolled our $(n - 1)$ -dimensional clustering along each spatial dimension of the cluster. Each of these unrolling operation is then called a *serialization*. The clustering process hence consists of a sequence of (up to) p *serializations*, the order in which these *serializations* are performed determines the resulting cluster schedule and the amount of local memory in each PE.

As opposed to the tiling transformation, clustering has a large impact on the register requirements of the PE architecture. The architecture of a PE *serialized* along a given processor axis i can be automatically obtained using the following transformation rules: (i) all temporal registers present in the architecture before *serialization* are duplicated by factor σ_i . (ii) feed back loops plus an associated multiplexer are created for all spatial registers along processor axis i .

If we assume that the *serializations* transformations are executed in the lexicographical order of the processor space

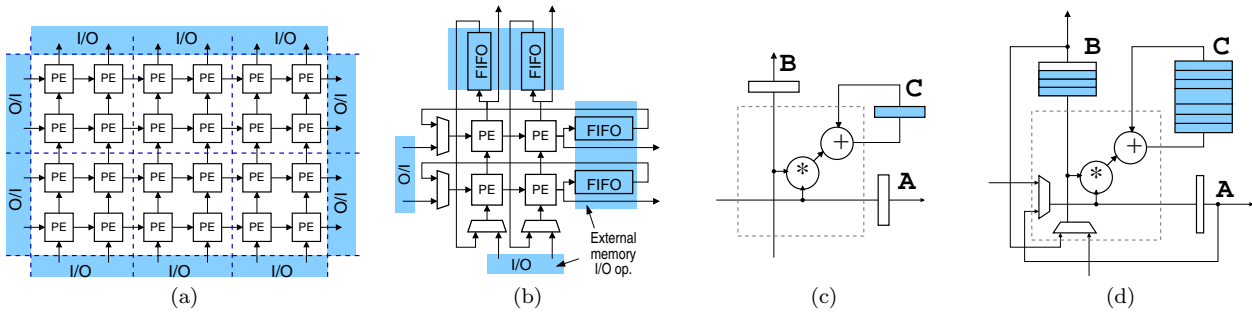


Figure 1: Parallelization process and post optimizations: (a) original array topology (b) tiled array architecture (c) original PE architecture (d) clustered (serialized) PE architecture.

base vector¹, the clustered PE architecture register usage can be obtained with the following transformation rules: (i) every initial temporal register is transformed into a $\prod_{i=1}^p \sigma_i$ deep shift register and (ii) every spatial register along processor axis i^{th} is transformed into a $\prod_{j=i+1}^p \sigma_i$ deep shift register.

2.4 Combining Tiling and Clustering

We now combine the two transformations as follows: we first select a spatial tiling (according to some criterion, for example the tile I/O volume) and derive its associated processor sub-array of $n_p = \det |\Omega|$ PE's. This processor array is then implemented on the target architecture after a *clustering* step represented by a transformation matrix Γ . The resulting processor array effective size is then given by $n_p = \det |\Omega| (\det |\Gamma|)^{-1}$, and the whole loop nest computation is executed as a succession of P passes, where P is the number of tiles on the tiled spatial domain, on an n_p PE processor array slowed down by a factor $\det |\Gamma|$. In spite of the restrictions we have imposed, there are a number of design parameters which can affect the final architecture.

3. HIGH-LEVEL POWER MODELING

Since there is a fairly large design space (we have $2n - 2$ independent partitioning parameters), we seek a high-level model for power dissipation of the derived system. Our target system architecture (shown in figure 2) consists of (i) an FPGA chip which implements the parallel processor array (and its associated controller), and (ii) one (or several) SRAM or DRAM memory banks containing the application data set.

We hence want to provide a high-level analytical model for (i) the energy dissipated by the individual PE's of the array (which is the sum of its data path and registers/memory), and (ii) the energy dissipated by the memory that stores principally the inter-tile data

3.1 FPGA core energy dissipation

The processor array architecture consists of two distinct parts: the array the processing logic and the array and I/O control logic. Since the power behavior of this latter one is difficult to characterize, and since it does not contribute significantly to the global energy bill, it will not be considered in our model.

¹Note that this does not affect the generality of our approach since by permuting rows in the matrixes Γ and π , it is possible to reorder these processor space axes.

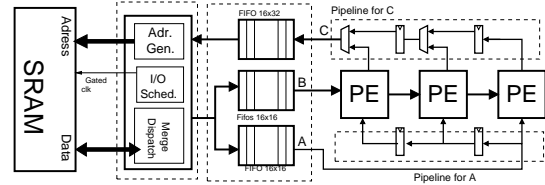


Figure 2: A view of the target architecture

Clearly, the energy dissipation of the PE array depends directly on (i) the PE architecture (which is affected by *serialization* parameters) and (ii) the number of PE's effectively implemented in the array (affected by *tiling* and *serialization*). Besides, it also makes sense to consider that all PE's implemented have the same energy behavior. The task is hence to provide an energy model for the PE, which should be parameterized by the partitioning parameters. To do so we will make the distinction between the energy dissipated by the functional data-path and the PE local memory.

Data-path energy model

The data-path consists of all combinational operators involved in the loop body, and remains unaffected by the partitioning transformation (with the exception of the muxes induced by the clustering transformation). We will hence estimate the whole data-path energy behavior using *activity power sensitive model* for each operator [1], where the energy dissipated by an operator is dependent on (i) its functionality and implementation, (ii) the activity statistics for its input pins. For each operator, its energy dissipated per clock cycle is then written as a third order polynomial function written as $E_{op} = k_0 + k_1 a_{op} + k_2 a_{op}^2 + k_3 a_{op}^3$, where k_i are the macro-model parameters, and a_{op} is the average activity at the operator input pins. The average energy dissipation per cycle associated to the combinational datapath is then written as $E_{dp} = \sum_{i=1}^{n_{op}} E_{op,i}$ when n_{op} is the number of operator in the data-path.

Local memory energy model

Although the combinational data-path of a PE is not affected by the partitioning transformations, its local memory is. The problem is hence to quantify the energy as a function of the *serialization* parameters. The main difficulty lies in the fact that, especially in recent FPGA families, there are many types of memory resources available for implementing the shift registers associated with our processor arrays.

We therefore model the energy behavior of the PE memory using a parametric macro-model which is a function of (i) the FIFO width and depth and (ii) the temporal correlation between input data. Since each shift register primitive can emulate up to a 16×1 FIFO, we have $E_m = k_0(1+k_1a_m)w \lceil \frac{d}{16} \rceil$ with k_i being the model parameters, w and d the FIFO width and depth and a_m the temporal correlation between the FIFO input data.

The problem is hence to determine the size (width and depth) along with the input data temporal correlation of all FIFO's present in the PE architecture. Let l_k denote the register count (in bits) associated with the spatial registers of the k^{th} processor space axis (l_0 denoting the number of temporal registers). Using the results of Section 2 describing the number of FIFO's introduced by serialization, we have

$$E_m = \sum_{i=0}^p k_0(1+k_1a_m)l_i \left[\frac{1}{16} \prod_{j=i+1}^p \sigma_j \right]$$

Let us now apply this model to our example. Let σ_1 and σ_2 denote the *serialization* factors along each of the two dimensions of our 2-D array. Since the local memory consists of a 32×1 -bit temporal register ($l_0 = 32$), and two 16×1 -bit spatial registers ($l_1 = l_2 = 16$), a $\sigma_1 \times \sigma_2$ *serialization* will yield an energy consumption of

$$E_m = k_0(1+k_0a_C)32 \lceil \frac{\sigma_1\sigma_2}{16} \rceil + k_0(1+k_1a_A)16 \lceil \frac{\sigma_1}{16} \rceil$$

Total core energy

We will first make the following approximation: when a processor is idle (has no useful computation to perform during a given clock cycle), we will consider that it does not contribute the the energy dissipation (this can be ensured through the use of clock gating techniques).

The energy dissipated by the processor array during the execution of the entire computation is thus approximated by the product of the average energy cost per iteration for a single PE by the iteration volume, V (i.e., the volume of the iteration domain of the original loop) and is given by $E_{core} = V(E_m + E_{dp})$.

Note that this formula is independent of the chosen level of parallelism, the only contributing parameter being the *serialization* parameters and the PE initial characteristics.

Partitioning and activity factor

By using *serialization*, computations within a PE are reorganized both in space and time, and the effective activity factor within the data-path is actually dependent upon the serialization parameters.

Since our PE energy model relies extensively on these activity factor, we must estimate their values for all set of serialization parameters. This step is, of course, likely to induce relatively long simulation time, but can be done directly from a high-level specification language (such a C) since it does not require a precise knowledge of the data-path structure.

3.2 Modeling I/O operation energy dissipation

Any off-chip I/O operation involves an electrical activity from both the FPGA pins and the memory chip, therefore the more I/O operations are required, the larger the total

dissipated energy. We can model the dynamic energy dissipation associated with a I/O operation as:

$$E_{i_o} = V_r k_{r,0}(1+k_{r,1}a_r) + V_w k_{w,0}(1+k_{w,1}a_w)$$

where $k_{r,0}$, $k_{w,0}$ and $k_{r,1}$, $k_{w,1}$ are parameters dependent the memory chip and FPGA technology and V_r and V_w the total number of I/O read and write operations. Hence, a way to reduce the total power dissipation is to reduce the global I/O volume exchange which is dependent upon (i) the loop nest characteristics (domain size, number and width of the variables, etc.) and (ii) the tiling transformation parameters.

Our first task is hence to quantify, knowing the loop nest and tiling characteristics, the I/O access volume associated with a tile. Then, knowing the number of tiles, we can easily determine the total I/O volume. This tile I/O volume (also known as tile footprint [2]) is determined by the data dependence vectors: when we tile our space-time domain, the space time projected dependence vectors split in two categories.

- *Internal dependences*: both "endpoints" of the data dependence vectors belong to the same tile. These correspond to local communication between PE's in a tile (or sub-array).
- *External dependences*: the endpoints of the data dependence vector belong to two different tiles. This corresponds to an external I/O access to the FIFO memory banks

The main difficulty in estimating this foot-print lies in the fact that the same data can be reused at several iterations. Argawal et al. have proposed a mathematical framework to approximate a tile foot-print for tiles with affine dependencies [2]. Since our loops have only uniform dependences, we can use an much simpler formulation and approximate the tile foot-print as follows: given a variable A with k dependence vector \vec{d}_k , one can express its associated spread vector a which corresponds to the maximum relative offset of these vector in each dimension.

For example, assuming $\vec{d}_0 = [0 \ 1 \ 6]$ and $\vec{d}_1 = [2 \ 5 \ 3]$ leads us to the spread vector $\vec{s} = [2 \ 4 \ 3]$. The tile foot-print can then be approximated by $V_A = \sum_{i=1}^n \det |\Omega_{i-\vec{s}}|$ where $\Omega_{i-\vec{s}}$ is the matrix obtained by replacing the i^{th} row of the tiling matrix Ω by the spread vector \vec{s} .

Argawal et al., have shown that though this value is an approximation, the error remains small in most cases. The complete tile foot-print is thus the sum of the foot-prints of each of the loop variables, and since Ω is a diagonal matrix with components ω_i , this volume can be written as

$$v_{in} = \sum_{k=0}^{n_{in}} l_k \left(\sum_{i=0}^n s_j \prod_{j=1, i \neq j}^n \omega_j \right) \quad (1)$$

$$v_{out} = \sum_{k=0}^{n_{out}} l_k \left(\sum_{i=0}^n s_j \prod_{j=1, i \neq j}^n \omega_j \right) \quad (2)$$

The energy dissipation associated with I/O operations for a whole tile execution is thus

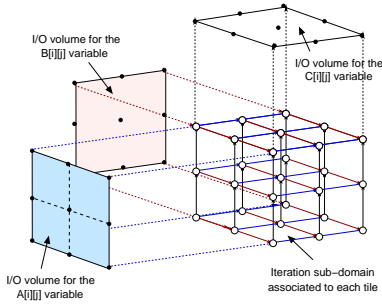


Figure 3: I/O volume (matrix product)

$$E_{io} = c_{in} \sum_{k=0}^{n_{in}} l_k \left(\sum_{i=0}^n s_j \prod_{j=1, i \neq j}^n \omega_j \right) + c_{out} \sum_{k=0}^{n_{out}} l_k \left(\sum_{i=0}^n s_j \prod_{j=1, i \neq j}^n \omega_j \right)$$

Let us estimate the I/O volume associated with the tiled version of our matrix multiplication example. As seen in 2.1, two input array variables (A and B) are accessed in the loop body. Their associated spread vector are $\vec{s}_A = [1 \ 0 \ 0]$ and $\vec{s}_B = [0 \ 1 \ 0]$, since it is a 16bits variable, we can estimate its footprint as $V_{in} = 2N\omega_1 + 2N\omega_2$ bytes. The spread vector associated with C (a 32 bits variable) can be written as $\vec{s}_C = [0 \ 0 \ 1]$, leading to a footprint of $V_{res} = 2\omega_1\omega_2$. The total tile I/O volume can then be written as $V_{io} = 2P.(\omega_1\omega_2 + 2N\omega_1 + 2N\omega_2)$ where P is the number of passes associated with the chosen tiling strategy.

3.3 Putting it all together

The global energy dissipation being the sum of the I/O energy dissipation and the FPGA core energy dissipation for a whole loop nest execution, it can be written as $E = n_{tiles}(E_{core} + E_{io})$.

4. EXPERIMENTAL VALIDATION

In this section we apply our model to our matrix multiplication example. We first give a brief overview of our target experimental platform. Next we describe how we determined some the architecture model constants. We then formulate the complete energy dissipation model for our example, and compare the predicted results to those measured experimentally.

Our experimental platform is a Spyder X2 FPGA emulation Board [8]. It consists of a Xilinx Virtex FPGA with two 256kx32 SRAM memory banks. For the sake of simplicity, our results reported here deal with a simplified architecture for which we choose to set $\sigma_j = \omega_j$ and $\sigma_j = \omega_j$ (i.e., an array with a single processor). This allows a clearer view of the trade-off between local memory and I/O regarding power dissipation. Given our hypothesis that all array PE's exhibit the same power behavior, this does not affect the relevancy of the results. Therefore there only remain two design parameters which are ω_i and ω_j .

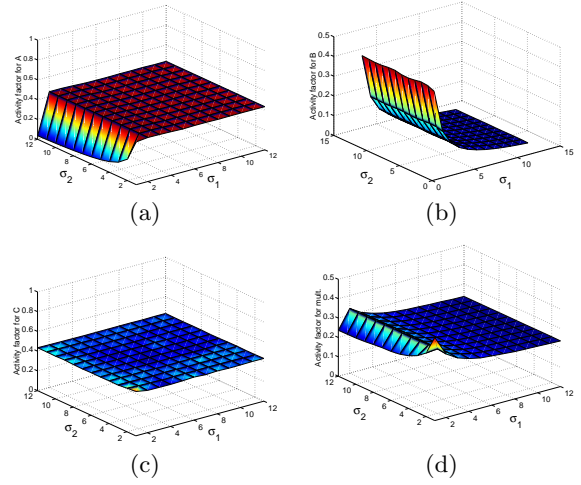


Figure 4: Activity factor for all variable in the PE datapath, as a function of the serialization parameters : see sub-figure (a), (b) for A, and (c) for C. Sub-figure (d) shows activity on the multiplier input port (given by $a_{mul} = \frac{1}{2}(a_A + a_B)$).

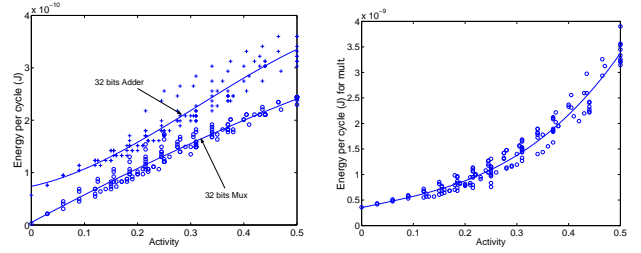


Figure 5: Combinational Macro-model calibration

Computing the activity factor

Following the principle described in 3.1, we have determined through simulation the activity factors associated to all variables present in the loop-body. The results obtained for these simulations are given in figure 4, and show that the activity factor within the datapath can be strongly influenced by the serialization parameters (see in particular sub-figure 4.d)

Calibrating the Macro-models

One of the main issues with such a high-level model is to determine the values of all the constant parameters associated to our macro-models (combinational operators, local memory and I/O operations).

Due to space constraints, we do not describe the details of our calibration here. The idea is simply to estimate the parameter values from a set of benchmark designs, with different characteristics: size, number of I/O, memory size and signal statistics. From these experimental data, we then estimate the parameter values by a least-squares fit. The results for this calibration step are shown in figures 5, 6 and 7.

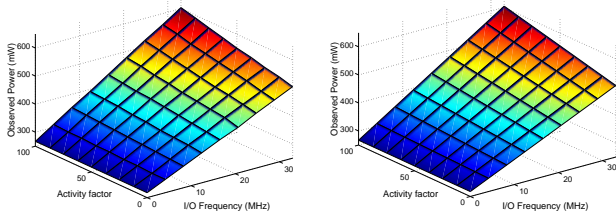


Figure 6: I/O Macro-model calibration (for write op.) observed vs macro-model

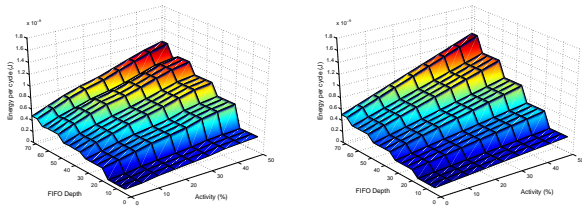


Figure 7: Local memory (FIFO) energy as a function of its input port activity and its depth : observed (a) vs macro-model (b)

Validating the whole model

The total energy bill for the execution of the whole loop on the architecture is given in figure 8. As expected, we observe the existence of a trade-off between the contribution of the external I/O accesses (that dominates for small tile size) and local memory (that dominates for large tiles).

The difference between the predicted and observed values is always less than 25% (worst case), with an RMS error of 12.7%. Moreover, the error is apparently not correlated to the partitioning parameters, which suggests that our model is relatively consistent. As a comparison, the only commercial tool existing for these devices, which perform the estimation from a fully placed and routed design, shows error varying between 5% and 10%.

Regarding speed, we managed to explore the whole design space in less than 1 hour (the most time consuming part being the determination of the activity factor within the PE datapath. Direct exploration (which means placing and routing all design instances, and manually measuring power dissipation) represents approximately 500 hours of computation time (for place and route) on a recent workstation (using a low-level power estimator such as Xpower, would require twice as much computing time).

5. CONCLUSION

In this paper, we have provided a high level power modeling analytical model for the implementation of partitioned processor arrays on FPGAs based system. This model was validated experimentally leading to reasonable error values in the estimate.

Since both theoretical and experimental results indicate the existence of a trade off between the global I/O volume and the PE local memory resource usage, and since the impact of each of these characteristics can be determined directly from the partitioning parameters, the next step would be to provide some analytical solution to the

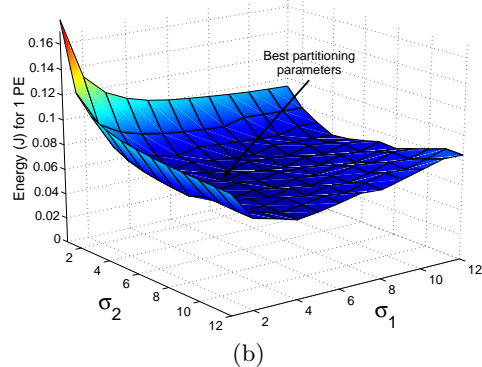
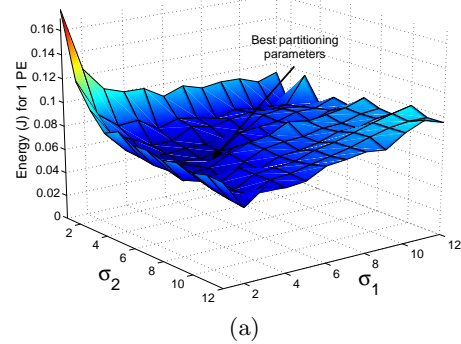


Figure 8: Whole loop energy as a function of partitioning parameters: observed (a) vs predicted (b)

power/partitioning optimization problem. This the the direction of our ongoing work.

6. REFERENCES

- [1] S. Dey A. Raghunathan, N. K. Jha. *High-Level Power Analysis and Optimization*. Kluwer Academic Publishers, 1998.
- [2] A. Agarwal et al. Automatic partitioning of parallel loops and data arrays for distributed shared-memory multiprocessors. In *IEEE Transactions on Parallel and Distributed Systems*, 1995.
- [3] F. Catthoor et al. *Custom Memory Management Methodology*. Kluwer Academic Publishers, 1998.
- [4] U. Eckhardt and R. Merker. Co-Partitioning - A Method for Hardware/Software design for scalable Systolic Arrays. In *Reconfigurable Architectures, ITPress*, 1997.
- [5] A. Darte et al. A Constructive Solution to the Juggling Problem. In *International Conference on Application Specific Processor Arrays (ASAP)*, 2000.
- [6] R. Schreiber et al. High-level Synthesis of Non-programmable Hardware Accelerators. In *IEEE conference on Application Specific Array Processor*, 2000.
- [7] L. Thiele J. Teich and L. Zhang. Scheduling of Partitioned Regular Algorithms on Processor Arrays with Constrained Resources. In *International Conference on Application Specific Processor Arrays (ASAP)*, 1996.
- [8] K. Weiss et al. Power estimation approach for sram-based fpgas. In *IEEE Symposium of Field Programmable Gate Array*, 2000.