Combining Optimizations in Automated Low Power Design

Qiang Liu, Tim Todman and Wayne Luk
Department of Computing, Imperial College, London SW7 2AZ, UK
Email: {qiang.liu2, timothy.todman, w.luk}@imperial.ac.uk

***Abstract*—Starting from sequential programs, we present an approach combining data reuse, multi-level MapReduce, and pipelining to automatically find the most power-efficient designs that meet speed and area constraints in the design space on Field-Programmable Gate Arrays (FPGAs). This combined approach enables trade-offs in power, speed and area: we show 63% reduction in power can be achieved with 27% increase in execution time. Compared to the sequential designs, our approach yields designs with up to 158 times reduction in execution time. Moreover, for a given execution time, our combined approach generates designs using up to 1.4 times less power than those produced by the same optimizations applied separately and can also find solutions missed by separating the optimizations.**

I. Introduction

We aim to combine power and performance optimisation in a single approach. Given a sequential design description, our approach explores the design space, finding the most power-efficient design meeting user speed and area goals. This allows trade-offs in power consumption, speed and area. Although we use a simple power model, the approach is modular enough to allow a more accurate model to be substituted.

The approach combines three optimisation techniques: data reuse, multi-level MapReduce and pipelining. We identify connections between these techniques and formulate the low power design space exploration in a Geometric Programming (GP) [1] model, which uses the techniques to meet user goals with the lowest power cost. The approach also applies straightforward transformations to 1) transform the input to suit the GP model; and 2) apply further optimisations after exploring the design space based on the GP model.

We compile from C input to Handel-C output description; however, our approach can adapt to other descriptions.

The contributions of this work are:

- An approach automating the generation of the lowest power design meeting user area and speed goals in the design space using a proposed experimental power model;
- A GP model exploring the design space combining data reuse, multi-level MapReduce and pipelining; and
- Evaluation on three real applications, showing that power-efficient designs can be determined under different constraints, and up to 1.4 times power reduction over separate optimisations can be achieved.

II. Background

Various techniques for power optimisation have been investigated [2]. To our knowledge, there is no tool currently available that can perform these optimizations automatically.

In [3] code transformations from floating-point to fixed-point and polynomial approximation of arithmetic operations are automated to optimise system power consumption in an embedded system. Loop transformations such as loop peeling, loop fusion, *etc.*, have been used in [4]; results show that these transformations alone do not significantly impact power consumption of a microprocessor system. Matsumura *et al*. [5] propose a new memory structure for low power embedded system design with two partitions: dynamic power dominated and static power dominated. The problem is formulated as a nonlinear program and solved using a heuristic algorithm. Data reuse exploration for power optimisation in FPGA-based system is carried out in [6] and is formulated as a multi-choice knapsack problem. However, these approaches require pre-processing of input code to work well.

In this paper, we use multiple transformations to automate the generation of a low power design. Unlike previous approaches, our approach reduces system power by combining 1) straightforward transforms, such as loop merging, and 2) design space exploration of more complex optimizations: data reuse, pipelining and multi-level MapReduce. Our approach can be extended to support further transforms.

a) Data reuse: [7], by buffering frequently used data in local memories, it significantly reduces off-chip memory accesses and thus reduces off-chip power consumption for data dominated applications. Our approach involves on-chip scratch-pad memory buffers [8], resulting in multiple data reuse options for each array reference: when, where and which elements of each array are buffered on-chip. Different options have different impacts on system power consumption and require different on-chip memory resources. In addition, buffered data can distribute across multiple on-chip dual-port memory banks, increasing memory bandwidth. Data reuse is constrained by the on-chip memory size.

b) Pipelining: [9], [10], widely used to improve system throughput, can also reduce dynamic power, since pipelined circuits suffer fewer glitches [11]. Data dependence, memory and computation resources constrain the initiation interval of pipelining and thus the execution schedule.

c) MapReduce: [12], [13] is a technique widely used to improve parallelism of large-scale computations. It partitions the computation into two phases: first, the *Map* phase, in which the same computation is performed independently on multiple data elements; second, the *Reduce* phase, calculating the final

978-3-9810801-6-2/DATE10 © 2010 EDAA

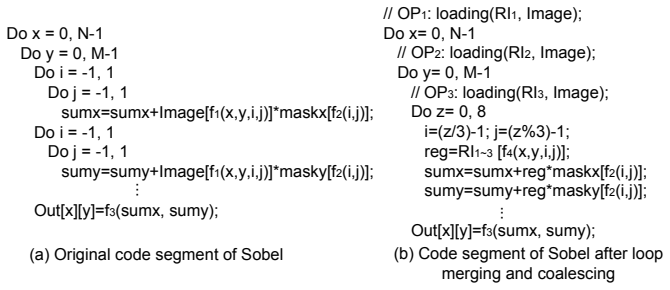

Fig. 1. Sobel code example.

result by combining the results of the map phase with an associative operator. We consider two-level MapReduce: *e.g.* in the Map phase of outer loops the iterations of inner loops are further MapReduced. This may allow designs to meet user speed goals not met by pipelining and data reuse. MapReduce is limited by memory bandwidth and computation resources.

These three techniques have not been used together before. In this paper, we show that combining these techniques allows a trade off between power consumption, speed and area.

III. Target Problem

We aim to automatically transform and optimise a sequential, but possibly inefficient design to minimize the system power consumption, while meeting user goals for speed and area. To achieve this, we combine three optimisation techniques: data reuse, pipelining, and multi-level MapReduce.

We observe that these optimisation techniques are interrelated. Memory resources constrain all three techniques. Data reuse, after duplicating buffered data across multiple memory banks, can improve memory bandwidth, benefiting pipelining and MapReduce. Computation resources also constrain pipelining and MapReduce. Therefore applying each technique individually to the input code may not lead to the most power-efficient design. Finally, we apply loop transformations, such as loop merging and coalescing, to automate the composition of the three techniques.

We illustrate the problem using the Sobel edge detection algorithm. Fig. 1 (a) shows the original code fragment, with four loop levels with two 2-level loops nested in the inner levels and two references to array `Image`, assumed stored in off-chip memory. Profiling shows that most elements of `Image` are accessed more than once and the access pattern of the two references to `Image` is the same; the only dependence between iterations of loops x and y and loops i and j is caused by the result `Out` and accumulation of `sumx` and `sumy`, respectively. We can thus add on-chip buffers for array `Image` and apply MapReduce so that the multiplications in different iterations execute in parallel in the map phase; the result output and accumulation execute in the reduce phase.

We first merge the inner two loop nests into one with i and j and then coalesce them to loop z (Fig. 1 (b)), simplifying the loop structure. Next, scalar replacement removes one reference to array `Image`.

Following the data reuse approach [8], there are three data reuse options (OP_1–OP_3) for array `Image`, as shown in Fig. 1 (b). Each option OP_j introduces an on-chip buffer RI_j, loads data from the off-chip memory into RI_j at a loop level and replaces the array reference `Image` with RI_j inside the innermost loop. If the image size is 144×176 pixels (8 bits), the on-chip memory required and number of off-chip memory accesses of options OP_1, OP_2 and OP_3 are (202752 bits, 25344), (4232 bits, 76032) and (72 bits, 228096), respectively. Larger on-chip buffers lead to fewer off-chip memory accesses and thus lower off-chip power consumption; however, larger on-chip buffers lead to higher on-chip power consumption. Our approach finds which data reuse option gives the lowest system power consumption.

We assume the off-chip memory has single access port. After introducing buffers in on-chip memory banks and duplicating data over multiple banks, memory bandwidth increases, allowing MapReduce to apply. Since all three loops in Fig. 1 (b) are parallelizable, the data reuse options govern which loop to partition. For example, if option OP_2 is chosen, only loops y and z can be parallelized, because the off-chip memory accesses for loading RI_2 exist in loop x. Furthermore, since the buffered data are duplicated to increase memory bandwidth, option OP_1 requires a large buffer, decreasing the number of duplications, given the fixed on-chip memory space, and thus decreasing the number of parallel partitions of the loop nest. It is also unclear which combination of data reuse options and the number of parallel partitions of the loop nest leads to a low power design with the required speed.

We simplify the problem, by only pipelining the innermost loop. That is, either only the innermost loop z is pipelined, or it is first partitioned into segments with all iterations in a segment executing in parallel and then the sequential segments execute in pipeline. Both MapReduce and pipelining are constrained by computation resources. In the Sobel example in Fig. 1 (b), if only two multipliers are available, then loop z could have two design options: 1) parallelize loop z into five segments where two iterations execute in parallel and then pipeline the five segments with initiation interval two; 2) pipeline loop z with initiation interval 1. The former could be faster, but consume more power in parallelized logic, while the latter could be slower but power efficient.

Combining options of data reuse, pipelining and MapReduce can result in a large number of design options. In our approach, we use a geometric programming (GP) model to automatically explore this large design space.

IV. Proposed Approach

A. Overview

Fig. 2 shows our approach. The inputs are the initial, sequential design, and the user goals: speed and target hardware platform. We first apply straightforward transformations to turn the input code into a form expected by the analysis toolbox; this simplifies the toolbox. Next, the analysis toolbox maps the regular loop iteration space in the code onto a

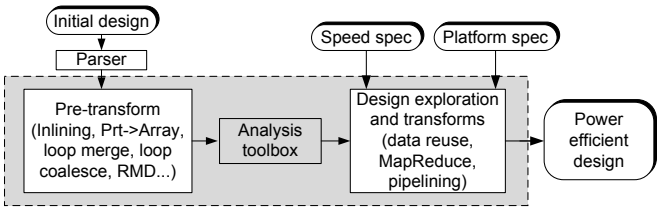

Fig. 2. Overview of the proposed approach.

polytope space and determines data dependencies and memory access patterns in the loops, for design space exploration.

First, the transformations enable code analysis and design exploration. The transforms shown in Fig. 2, such as function inlining and pointer dereference, reveal the data flow of the input code. In the Sobel example in Fig. 1, loop merging removes one reference to array `Image` and loop coalescing reduces the number of loops, and thus reducing the number of variables to be determined in design exploration.

The design exploration with combined optimisation techniques is formulated as an optimisation problem with the objective and constraint functions, whose globally optimal solution gives the optimal design option. In this paper, the design objective is low power, and user goals of speed and platform hardware resources (area) form the constraints.

Finally, the output code is transformed according to the solution. The final output design meets the design goals, maintaining initial design behavior.

The next section shows the formulation of the low power design exploration with data reuse, MapReduce and pipelining.

B. Formulation of design exploration

Without loss of generality, our approach presented in this paper targets an FPGA-based platform with single-port off-chip SRAM, assuming that each assignment in the code takes one clock cycle. The formulation below can easily be extended to deal with multiple off-chip ports and multiple clock cycle execution of statements.

For a loop with N levels $(I_1, \ldots, I_N)$, where I_1 is the outermost loop and loop I_N is the innermost loop, and R array references A_i $(1 \leq i \leq R)$, our problem is to choose: data reuse options OP_{ij} for reference A_i; the number of parallel partitions k_l of loop l with L_l iterations; and the initiation interval ii of the pipeline of loop N, so that the transformed design is the most power-efficient meeting design goals. This corresponds to the solution of the following problem $\mathcal{P}$:

$$\begin{aligned} \min_{\substack{\rho_{ij}\in\{1,2\} \\ 1\leq k_l \leq L_l \\ 1 \leq ii}} \ & \mathbb{P}(\rho_{ij}, k_l, ii) \\ \text{subject to} \ & \mathbb{S}(\rho_{ij}, k_l, ii) \leq T \qquad (\mathcal{P}) \\ & \mathbb{R}_f(\rho_{ij}, k_l, ii) \leq Res_f \\ \text{for} \ & \forall \ \ i = 1, \ldots, R, \quad j = 1, \ldots, E_i, \\ & l = 1, \ldots, N, \quad f \in F \end{aligned}$$

where T and Res_f are, respectively, the design execution time specification and availability of each f of F kinds of resource.

As the on-chip embedded DSP and RAM blocks are scarce, we prioritize saving DSP blocks and memory blocks. Boolean variable $\rho_{ij} = 2$ means data reuse option OP_{ij} is selected for A_i; otherwise $\rho_{ij} = 1$. E_i is the number of data reuse options of array A_i. The system power consumption $\mathbb{P}$, the execution time $\mathbb{S}$ and the resource utilization $\mathbb{R}$ are formulated as follows.

Power model. Since the objective function includes the power model, at this stage the model needs only to distinguish different design options. Also, as static power is constant on the same platform for different design options, we only consider dynamic power; in the rest of this paper power means dynamic power. We call the power consumed in off-chip memory the off-chip power consumption, which is governed by the off-chip memory access frequency. In contrast to [6], the on-chip power must consider parallelization caused by MapReduce. Therefore,

$$\begin{aligned} \mathbb{P} &= \mathbb{P}_{off} + \mathbb{P}_{on} \\ &= V_{dd} \times (I_{operating} - I_{sleep}) \times \frac{\#off\_accesses}{cyc} \\ &\quad +(P_1 \times \#off\_accesses + P_2 \times \#par + P_3 \times \#dsp \\ &\quad +P_4 \times \#bram \times BW + P_5) \times freq \end{aligned} \tag{1}$$

where V_{dd}, $I_{operating}$ and I_{sleep} are the working voltage, working current and standby current of the off-chip SRAM, found in the SRAM datasheet. #*off_accesses* is the number of off-chip memory accesses:

$$\#off\_accesses = \sum_{i=1}^{R} \prod_{j=1}^{E_i} \rho_{ij}^{\ log_2 C_{ij}} \tag{2}$$

$$\rho_{ij} \in \{1, 2\}, 1 \leq j \leq E_i, 1 \leq i \leq R \tag{3}$$

$$(E_i + 1)^{-1} \sum_{j=1}^{E_i} (\rho_{ij}) = 1, 1 \leq i \leq R \tag{4}$$

where C_{ij} is the number of cycles to load the on-chip buffer in data reuse option OP_{ij}. Equation (4) ensures exactly one data reuse option is chosen for each array reference. The execution time model gives the number of execution cycles, cyc.

Parameters P_1–P_5 characterize on-chip power consumption: IO, parallel logic, computation resources, on-chip RAM blocks and other sequential logic, respectively; these components vary across different design options. The parameters are constant for each input design and can be obtained by experimenting with designs on the target platform.

$\#par$ is the total number of partitions of the loop nest:

$$\#par = \prod_{l=1}^{N} k_l \tag{5}$$

$$1 \leq k_l \leq L_l, 1 \leq l \leq N \tag{6}$$

$$k_l \prod_{j=1}^{l} \rho_{ij}^{\ -log_2 L_l} \leq 1, 1 \leq l \leq N, 1 \leq j \leq E_i, 1 \leq i \leq R \tag{7}$$

where loop l is partitioned into k_l parallel segments. Inequality (7) links variables ρ_{ij} and k_l; loop l can only be parallelized

if the array references within its body have been buffered on-chip prior to loop execution. Section III discusses an example involving Sobel edge detection.

Similarly, the computation resources required are:

$$\#dsp = \prod_{l=1}^{N} k_l \times x_{dsp} \tag{8}$$

$$1 \leq x_{dsp} \leq Res_{dsp} \tag{9}$$

where x_{dsp} is the number of on-chip DSP blocks used in one loop nest partition. For simplicity, we assume that the computation is inside the innermost loop; the formulation can easily be extended to cover more general cases.

$\#bram$ is the total number of on-chip RAM blocks used:

$$\#bram = d \sum_{i=1}^{R} \prod_{j=1}^{E_i} \rho_{ij}{}^{log_2 B_{ij}} \tag{10}$$

where B_{ij} is the number of on-chip RAM blocks required by data reuse option OP_{ij}, and d is the number of duplications of the buffered data used to increase the memory bandwidth. We build a 2-level on-chip memory hierarchy: on-chip SRAM and registers. When the outer loops of the loop nest are parallelized, each on-chip SRAM bank is the local memory for each loop partition. Then, the inner loops within each outer loop partition are further parallelized; for these inner loop partitions, each on-chip SRAM bank is the globally shared memory and registers are used as local memory. Therefore the number of data duplications in on-chip SRAM is:

$$d = \lceil \frac{1}{2} \prod_{l=1}^{W_r} k_l \rceil \tag{11}$$

where W_r is the loop level containing the reduce statement, and the loops from the outermost loop to loop W_r are MapReduced. For example, in the Sobel code shown in Fig. 1 (b), $W_r = 2$ because the result output `Out` is within the second loop y. The factor of $\frac{1}{2}$ is because each on-chip dual-port SRAM bank is accessed by two parallel partitions. BW is the bitwidth of the on-chip memory.

Finally, $freq$ is the design clock frequency. Formulation (1) shows its effect on the five on-chip power components.

Execution time can be expressed as $\mathbb{S} = cyc/freq$. Therefore, we calculate cyc. For simplicity, we only apply combined pipelining and MapReduce to the innermost loop and MapReduce to outer loop levels; the formulation can however be generalized. The execution cycle count of the design comprises four parts:

$$cyc = cyc_s + cyc_{in} + cyc_r + \#off\_accesses$$

The number of cycles taken by statements outside the innermost loop cyc_s is given by Equality (12), where statement s among S statements is in loop level W_s and one partition of

$$cyc_s = \sum_{s=1}^{S} \prod_{l=1}^{Ws} v_l \tag{12}$$

$$L_l k_l^{-1} v_l^{-1} \leq 1, 1 \leq l \leq N \tag{13}$$

loop l has $v_l = \lceil L_l/k_l \rceil$ iterations defined in Inequalities (13).

The number of cycles taken by statements inside the innermost loop after two-level MapReduce and pipelining is:

$$\begin{aligned} cyc_{in} \quad = \quad & \prod_{l=1}^{N-1} v_l (v_N \times ii + C_{data} + \sum_{i=1}^{I} d_i \\ & + \lceil \log_2 k_N \rceil + notFull) \end{aligned} \tag{14}$$

$$W_{dsp} \times x_{dsp}^{-1} \times ii^{-1} \leq 1 \tag{15}$$

$$RecII \times ii^{-1} \leq 1 \tag{16}$$

$$BandW \times k_N \times M_b^{-1} \times ii^{-1} + notAlign \times ii^{-1} \leq 1 \tag{17}$$

$$R_{idsp} \times x_{dsp}^{-1} \times d_i^{-1}, 1 \leq i \leq I \tag{18}$$

where ii is the pipelining initiation interval, constrained by the computation resources in Inequality (15), data dependence in Inequality (16) and memory bandwidth in Inequality (17); C_{data} is the number of cycles to read one datum from on-chip RAM to registers. $\sum_{i=1}^{I} d_i$ is the number of computation cycles. There may be I ($I \geq 1$) computation levels in the data flow graph of the input code and the computation cycles of each loop iteration comprises the execution cycles of every level. The execution cycles d_i of computation level i are given by R_{idsp}, the resource DSPs required in level i and the allocated resource x_{dsp}, defined in Inequalities (18). $\lceil \log_2 k_N \rceil$ is the number of cycles for the reduce phase using a tree structure to reduce computation results, and the boolean parameter $notFull = 1$ when the innermost loop is not fully parallelized and one cycle is needed to accumulate results from different partitions; otherwise $notFull = 0$. W_{dsp} in Inequality (15) and $BandW$ in Inequality (17) are defined as the number of computation resource DSP and the memory bandwidth required per iteration of the innermost loop. M_b is the memory bandwidth available in accessing on-chip RAM. The boolean parameter $notAlign = 1$ if data are unaligned between storage and computation, as one extra access may need to obtain requested data; otherwise $notAlign = 0$.

Lastly, the number of cycles cyc_r taken by the reduce phase of the outer loop MapReduce is:

$$cyc_r = \prod_{l=1}^{W_r} v_l k_l \text{ or } cyc_r = \prod_{l=1}^{W_r} v_l \log_2 \prod_{l=1}^{W_r} k_l. \tag{19}$$

These two expressions for cyc_r correspond respectively to using a linear structure or a tree structure in the reduce phase.

Resource utilization is measured mainly in the on-chip memory and embedded DSP blocks. The on-chip DSP block and RAM utilization are given in Equations (8) and (10), so the on-chip resource constraint $\mathbb{R}$ is defined in Inequalities (20) and (21), where Rec_{dsp} and Rec_{ram} are the number of on-chip DSP blocks and RAM blocks available in the target platform.

$$\prod_{l=1}^{N} k_l \times x_{dsp} \leq Rec_{dsp} \tag{20}$$

$$d \sum_{i=1}^{R} \prod_{j=1}^{E_i} \rho_{ij}{}^{log_2 B_{ij}} \leq Rec_{ram} \tag{21}$$

TABLE I
THE DETAILS OF THREE KERNELS.

Kernel	k_l	Reference	Option	B_{ij}	C_{ij}
Sobel	$1 \le k_1 \le 144$	Image	OP_{11}	13	25344
	$1 \le k_2 \le 176$		OP_{12}	1	76032
	$1 \le k_3, k_4 \le 3$		OP_{13}	1	228096
ME	$k_1 = 1$	current	OP_{11}	13	25344
	$1 \le k_2 \le 9$		OP_{12}	1	25344
	$1 \le k_3, k_4 \le 16$	previous	OP_{21}	13	25344
MAT64	$1 \le k_1 \le 64$	A	OP_{11}	2	4096
	$1 \le k_2 \le 64$		OP_{12}	1	4096
	$1 \le k_3 \le 64$	B	OP_{21}	2	4096

Finally, we substitute $\mathbb{P}$, $\mathbb{S}$ and $\mathbb{R}$ in the problem $\mathcal{P}$ with the formulations above. All low case variables, except for $freq$ which is a real number, are integers, while capitals are compile-time constants. We observe that the only item making the problem $\mathcal{P}$ not a GP problem [1] is the logarithm in Equation (14). However, $\lceil log_2 k_N \rceil$ is constant in certain ranges of k_N. Therefore, the problem $\mathcal{P}$ can be seen as a piecewise mixed-integer geometric problem (MIGP) in different ranges of k_N. The number of subproblems increases logarithmically with k_N. The MIGP can be solved by a branch and bound approach using the GP solver as the lower bounding procedure.

V. EXPERIMENTAL RESULTS

We apply the framework to three kernels: multiplication of two 64×64 matrices (MAT64), one search path of the motion estimation (ME) algorithm [14] used in X264, and the Sobel edge detection algorithm (Sobel) [15]. The execution time, power and energy of the original design of these kernels with no optimization are shown in the captions of Figs. 3, 4 and 5. Table I shows details of these kernels. Our target platform is an FPGA-based system with off-chip SRAM having a single access port with two-cycle latency; all array references with input data are stored in off-chip SRAMs. The three kernels are implemented in Xilinx XC4VFX140 with 192 DSP48 and 552 dual-port RAM blocks. In this platform, we set the clock frequency range $1 \sim 100$ MHz and execution time constraints up to $158\times$ speedup over the original designs. For ME and Sobel the frame size is 144×176 pixels.

In our experiments, after initial transforms, the GP model (1)-(21) is applied to the three kernels to find data reuse options, MapReduce pattern and pipelining schemes, giving the lowest power design meeting the speed requirement. Solutions of the GP model under different speed specifications are shown in Figs. 3, 4 and 5 in downward-pointing triangles for the three kernels. Each design is represented by $(OP_{ij_{1 \le i \le R}}, k_{l_{1 \le l \le N}}, ii, freq)$, the GP solution. For example, in Fig. 3, the most power-efficient design for Sobel edge detection with execution time within 0.51 ms is $(OP_{11}, 4, 11, 1, 1, 100)$, *i.e.* the first data reuse option as shown in Table I is selected for the array `Image`, the first (outermost) loop is partitioned into 4 parallel segments, the second loop is partitioned into 11 parallel segments, the innermost loop without partitioning after merging is pipelined with initiation interval 1, and the clock frequency is 100 MHz. Similarly, the solution for MAT64 is $(OP_{12}, OP_{21}, 1, 2, 4, 1, 70)$ shown in Fig. 4, where the speed specification is 0.9 ms. We use the branch and bound algorithm in YALMIP [16] to solve the GP model described in Section IV; each design is generated within 2 minutes on average.

To verify the designs given by the GP model, we implement them on the target platform to obtain execution time and power consumption. The power values are estimated by Xilinx Power Estimator [17]; these results are shown by stars in Figs. 3, 4 and 5. We see that the downward-pointing triangle line and the star line have the same variation over different execution time constraints for the three kernels. This demonstrates that the power model used in our approach correctly determines the most power-efficient design under different execution time constraints. Moreover, all stars, except for the two in the middle of Fig. 3 which are 0.4% slower than the estimated designs, are on the left hand side of the corresponding downward-pointing triangles, showing that speed requirements are met. We also implement several possible designs for the three kernels, as shown by dots in Figs. 3, 4 and 5. These design options have been automatically removed by our design exploration model, and are all above or to the right of the power-efficient designs (the stars), *i.e.* they either consume more power or run slower. Also, note that power increases as execution time reduces (speed goes up), increasing rapidly at high speeds. Therefore, users can trade speed for power. For instance, in MAT64, increasing execution time by 27% (0.04 ms) reduces power consumption by 63% when the execution time constraint is below 0.2 ms.

Our approach combines data reuse, MapReduce and pipelining. Figs. 3, 4 and 5 (circle data points) show the advantage of combining optimisations over separate optimisations, where data reuse applies to reduce off-chip memory access and then MapReduce and pipelining are applied to increase speed. For Sobel, when the execution time constraint exceeds 0.6 ms, the designs proposed by our approach consume up to 1.4 times less power than the separate optimisations. For MAT64, in most cases the designs proposed by both approaches are the same, because the different data reuse options have the similar effects on system performance as shown in Table I. However, when the execution time constraint is less than 0.15 ms, the combined approach finds a solution missed by the separate approach. In the ME case, the two produce the same result, because the outermost loop cannot be MapReduced due to data dependence, *i.e* the dependence between data reuse and MapReduce does not exist. Therefore, for applications having data reuse options significantly varying and thus tightly interlinked with MapReduce and pipelining options, our combined approach is more beneficial. Moreover, many loop partitions do not guarantee the fastest design. For example, in Fig. 3, two non-optimal designs shown by dots in the up-left corner have 1584 and 528 parallel partitions, but are no faster than 0.54 ms and consume significant power.

The off-chip and on-chip dynamic power of the designs proposed by our approach are also shown in Figs. 3, 4 and 5. We observe that on-chip dynamic power increases as the

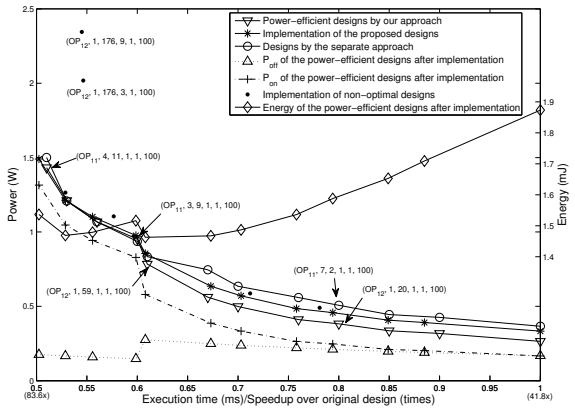

Fig. 3. Sobel results. Ori-design: time=41.8ms, power=0.08W, energy=67mJ.

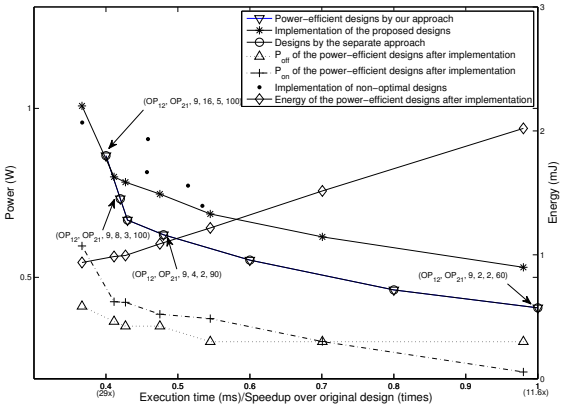

Fig. 5. ME results. Ori-design: time=11.6ms, power=0.37W, energy=22mJ.

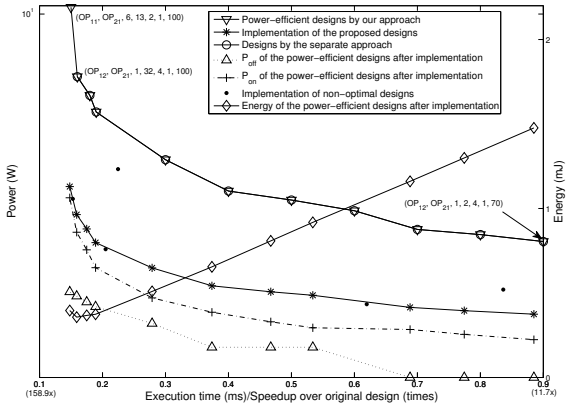

Fig. 4. MAT64 results. Ori-design: time=15.9ms, power=4W, energy=81mJ.

execution time constraint tightens. This is because more on-chip resources are used to parallelize computation and the clock frequency is increased. However, the off-chip dynamic power does not monotonously increase. For instance, in Fig. 3, the off-chip dynamic power decreases when the execution time constraint is tighter than 0.6 ms. This is because the data reuse option selected for the design has changed, as shown in Fig. 3. When the time constraint is less than 0.6 ms, the data reuse option with fewer off-chip accesses is chosen. We can also see some non-monotonic behavior in the off-chip dynamic power for MAT64 and ME. This validates our approach considering all factors which cause system dynamic power variations.

Finally, users can also trade energy for execution time. Energy reduces with execution time for ME but shows minima for Sobel and MAT64: sweet spots for battery operation.

VI. Conclusion

We present an optimization method combining multiple optimizations: data reuse, pipelining and multi-level MapReduce, together with transforms such as loop merging and function inlining. User inputs are a sequential program and speed and hardware resource constraints. Our method generates the lowest power design meeting those constraints. Using our approach, users can trade speed for power: 27% increase in execution time can reduce power consumption by 63%. Also, we show that designs generated by our combined approach use up to 1.4 times less power than those applying the three optimizations separately; furthermore, the combined approach finds solutions that are missed by the separate approach.

Future work includes supporting more applications and adding more optimization methods to our combined approach, such as previous work on data representation optimization.

References

[1] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2004.

[2] J. Lamoureux and W. Luk, "An overview of low-power techniques for field-programmable gate arrays," *NASA/ESA Conference on Adaptive Hardware and Systems*, vol. 0, pp. 338–345, 2008.

[3] A. Peymandoust *et al.*, "Low power embedded software optimization using symbolic algebra," in *DATE '02*. IEEE Computer Society, 2002, pp. 1052–1058.

[4] D. A. Ortiz and N. G. Santiago, "High-level optimization for low power consumption on microprocessor-based systems," in *MWSCAS 2007*. IEEE Computer Society, 2007, pp. 1265–1268.

[5] T. Matsumura *et al.*, "Simultaneous optimization of memory configuration and code allocation for low power embedded systems," in *GLSVLSI '08*. ACM, 2008, pp. 403–406.

[6] Q. Liu *et al.*, "Data-reuse exploration under an on-chip memory constraint for low-power FPGA-based systems," *IET Computers & Digital Techniques*, vol. 3, no. 3, pp. 235–246, 2009.

[7] F. Catthoor *et al.*, *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design*. Norwell, MA, USA: Kluwer, 1998.

[8] Q. Liu *et al.*, "Data reuse exploration for FPGA based platforms applied to the full search motion estimation algorithm," in *Proc. Int. Conf. on FPL*, Matrid, Spain, 2006, pp. 389–394.

[9] K. Turkington *et al.*, "Outer loop pipelining for application specific datapaths in FPGAs," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 16, no. 10, pp. 1268–1280, 2008.

[10] H. Rong *et al.*, "Single-dimension software pipelining for multi-dimensional loops," in *IEEE Proc. on CGO*, 2004, pp. 163–174.

[11] S. J. Wilton *et al.*, "The impact of pipelining on energy per operation in field-programmable gate arrays," in *FPL*. Springer, 2004, pp. 719–728.

[12] J. H. Yeung *et al.*, "Map-reduce as a programming model for custom computing machines," in *FCCM*, 2008, pp. 149–159.

[13] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *6th Symp. on OSDI*, December 2004, pp. 137–150.

[14] L. Merritt and R. Vanam, "Improved rate control and motion estimation for h.264 encoder," in *ICIP 2007*, 2007, pp. 309–312.

[15] http://www.pages.drexel.edu/∼weg22/edge.html, accessed 2006.

[16] J. Lfberg, "YALMIP : A toolbox for modeling and optimization in MATLAB," in *Proc. Conf. CACSD*, Taipei, Taiwan, 2004.

[17] http://www.xilinx.com, "Xilinx power estimator user guide," accessed 2009.