A Dynamic-Programming Algorithm for Reducing the Energy Consumption of Pipelined System-Level Streaming Applications

N. Liveris
Northwestern University
Evanston, IL 60208 USA

H. Zhou
Northwestern University
Evanston IL 60208 USA

P. Banerjee
HP Labs
Palo Alto, CA 94301 USA

Abstract

In this paper we present a System-Level technique for reducing energy consumption. The technique is applicable to pipelined applications represented as chain-structured graphs and targets the energy overhead of switching between active and sleep mode. The overhead is reduced by increasing the number of consecutive executions of the pipeline stages. The technique has no impact on the average throughput. We derive upper bounds on the number of consecutive executions and present a dynamic-programming algorithm that finds the optimal solution using these bounds. For specific cases we derive a quality metric that can be used to trade quality of the result for running-time.

1 Introduction

Synchronous Dataflow Graphs (SDFs) are considered a useful way to model Digital Signal Processing applications [1]. This is because in most cases the portions of DSP applications, where most of the execution-time is spent, can be described by processes or actors with constant rates of data consumption and production.

Energy consumption is one quality metric for digital integrated circuits. The main sources of energy consumption are dynamic and static power dissipation. Static or leakage power is expected to become the dominant power dissipation component for future technologies [8]. Therefore, techniques to reduce the leakage power are needed.

Work on leakage reduction at the higher levels of design has been focused on replacing cells or submodules of the design with ones with the same functionality but higher threshold voltage (e.g. [5]). Although these techniques can lead to significant reductions, they are not applicable to parts of the design that come as hard cores or when the available time slack changes, even with a low frequency, e.g. by the user of the system. In these cases, techniques are needed that are adaptive to environment changes and do not require resynthesis of IP cores. Such techniques include Dynamic Voltage Scaling [9], Adaptive Body Biasing [10], and Power Gating [8]. In this paper we focus on the latter technique.

With power gating, a hardware module is shut down when it is idle. This way the stand-by leakage of the module is reduced. The switching from active to sleep mode and back to active has an energy penalty caused mainly by the loading of the nodes to normal V_{dd} levels [8]. In this work we try to decrease energy consumption by reducing the number of times the mode switch occurs.

In this paper we try to find the number of consecutive iterations for each pipeline stage of a chain-structured SDF graph. This problem is similar to vectorization [7], but in our case instead of trying to maximize the consecutive number of executions, we try to maximize the energy savings taking into account the energy penalty paid by adding more buffers to each channel. Dynamic programming techniques have been used to determine a schedule for a chain-structured SDF, so that the memory requirements are minimized [3]. In our approach, the buffer requirements are increased, whenever this increase leads to a reduction of the total energy consumption.

The throughput of the application does not change after applying our method. Moreover, our method guarantees that any latency increase does not cause data loss. In general for streaming multimedia applications throughput constraints are important and less emphasis is put on latency [4]. Our technique is applicable only to streaming applications, for which a latency increase is acceptable.

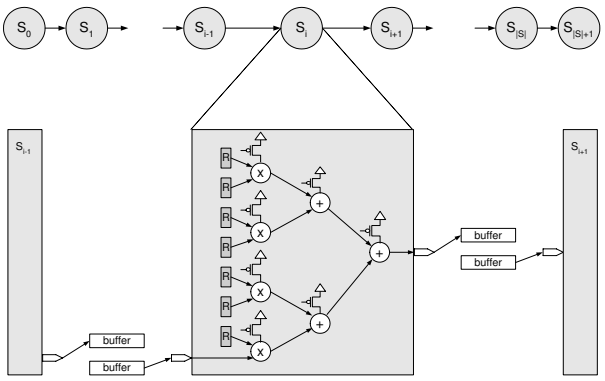

Figure 1: System structure. At the first level of hierarchy the system is a pipelined chain-structured graph. The processes (nodes) of the second level can be independently power gated. Cross edges between pipeline stages are implemented using buffers.

In Section 2 we explain the model we use to describe pipelined system-level applications. Section 3 defines the problem we try to tackle. In Section 4 the theoretical issues of the problem are addressed, while Section 5 describes an algorithm that can be used to solve it. Finally, in Sections 6 and 7 we present experimental results and draw conclusions.

Most proofs have been omitted due to space limitations. However, they are published in a report, which is available in our website [12].

2 Model Description

In this section we describe the model we use for system-level pipelined applications. Table 1 summarizes the definitions of the symbols used in this paper. In Figure 1 the structure of the model can be seen.

2.1 Chain-Structured SDFs

In an SDF $G=(V,E)$ each node represents a process and each edge a channel, in which the tail produces data and the head consumes data. We assume a global clock for the whole system. Functions $p: E \to \mathbb{N}$, $c: E \to \mathbb{N}$, and $w: E \to \mathbb{Z}_0^+$ represent the production, consumption rates, and the number of initial tokens (delays) of each channel.

In order for an SDF to be executable with bounded memory, the system $\Gamma\tilde{q}=0$ should have non-trivial solutions, where Γ is the topology matrix of G [1]. The vector with the minimum positive integers in the solution space, $\tilde{q}$, is called the repetitions vector and each entry represents the number of times the corresponding node should be executed during each complete cycle of the graph. An SDF is called consistent if it has a repetitions vector and the system does not deadlock [2].

978-1-4244-1922-7/08/$25.00 ©2008 IEEE

Symbol	Definition
q_s	number of executions (instances) of stage s in one complete cycle
$p(i,i+1)$	number of tokens produced on cross edge (i,i+1) as a result of one execution of stage s_i
$c(i,i+1)$	number of tokens of cross edge (i,i+1) consumed as a result of one execution of stage s_{i+1}
$w(i,i+1)$	number of initial tokens (delays) on cross edge (i,i+1)
$b(i,i+1)$	number of buffers on cross edge (i,i+1)
$l(G_s)$	execution time in cycles for each instance of a pipeline stage s
$l(v)$	the number of cycles process $v \in V_s$ must remain active during the execution of one instance of s
$E_{sm}(v)$	energy overhead for switching modes from active to sleep and back to active for process v
$\Delta P(v)$	power difference between active and sleep mode when process v is idle
L_{cc}	period of execution for a complete cycle of the pipeline (chain-structured SDF graph)
L_s	period of invocation for pipeline stage s, initially equal to $\frac{L_{cc}}{q_s}$
x_s	number of consecutive instance executions of pipeline stage s, initially equal to 1
ρ	quality metric of the solution, applicable only to unirate SDF graphs
$E_s(v)$	energy savings from a process v
$E_p(i,i+1)$	energy penalty on cross edge (i,i+1)
$E_t(\tilde{x})$	the total energy savings after subtracting the total energy penalty on the channels for a solution $\tilde{x}$
$\mathbb{N}$	the set of natural numbers
$\mathbb{Z}_0^+$	the set of non-negative integers ($\mathbb{N} \cup \{0\}$)

Table 1: Definition of Commonly Used Parameters

A proper subset of E in the graph may not have either a tail or a head. These are the input and output edges with which the SDF communicates with its environment.

In case all production and consumption rates are equal with 1, the graph is called a unirate SDF. Otherwise, it is called a multirate SDF. A unirate SDF has a repetition vector with all entries 1.

The subset of SDFs we are interested in can be represented in the first level of hierarchy as a chain-structured directed multi-graph $G = (S,E)$ [3] with nodes that are all executed in parallel. We define G as a graph with $|S|$ nodes, for which there are labels $s_1, s_2, ..., s_{|S|}$, such that each edge $e \in E$ can be directed only from s_i to s_{i+1} for any i. Therefore, there can be multiple edges between two nodes, but edges can only connect nodes, whose labels differ by one, in the direction from the smallest label to the greatest. We call these nodes *pipeline stages* or *stages* and we call the edges between pipeline stages *cross edges*.

Properties of hierarchical clustering of SDFs are described in [2]. In our case we assume the clustering has been done to satisfy an average throughput constraint for the graph and to minimize the cost of pipelining on cross edges. Here we assume that the data a stage consumes have to be available until the end of the stage's execution. Moreover, the memory to store the data produced by a stage should be available before the starting time of that stage.

All input edges of the application SDF become cross edges, whose head is s_1 and whose tail is stage s_0, which is external and we have no control over it. An external stage $s_{|S|+1}$ is defined for the output edges, as well. Each stage s is already synthesized and has an execution time of $l(s)$ cycles.

2.2 Processes

Each pipeline stage s can be represented by a directed graph $G_s = (V_s, E_s)$, where V_s is the set of processes and E_s is the set of edges (channels) between the processes.

Function $l : V \rightarrow \mathbb{N}$ returns the number of clock cycles process $v \in V_s$ must remain active during the execution of s.

We assume that when a process v is idle, it can be in an active and power-hungry or a sleep and power-efficient mode. The power difference is $\Delta P(v) = P_{acm} - P_{slm}$, where P_{acm} and P_{slm} are the power in active and sleep mode. $P_{ac2slm}(v)$ and $P_{sl2acm}(v)$ are the average power consumptions during switching modes and $t_{ac2slm}(v)$, $t_{sl2acm}(v)$ the time periods needed for the switching. Then if v does not switch mode the total energy dissipated in the slack time is:

$$E_{ac} = \Delta t \cdot P_{acm}$$

while if it is switched to sleep mode the total energy dissipated is:

$$E_{sl} = (\Delta t - t_{ac2slm} - t_{sl2acm}) \cdot P_{slm} + t_{ac2slm} \cdot P_{ac2slm} + t_{sl2acm} \cdot P_{sl2acm}$$

The energy savings for switching a node v from active to sleep mode during some time interval Δt, in which the process is idle, are

$$\begin{aligned} E_s(v) &= E_{ac}(v) - E_{sl}(v) \\ &= \Delta t \cdot (P_{acm}(v) - P_{slm}(v)) \\ &\quad - P_{ac2slm}(v) \cdot t_{ac2slm}(v) - P_{sl2acm}(v) \cdot t_{sl2acm}(v) \\ &\quad + P_{slm}(v) \cdot t_{ac2slm}(v) + P_{slm}(v) \cdot t_{sl2acm}(v) \\ &= \Delta t \cdot \Delta P(v) - E_{sm}(v) \end{aligned} \quad (1)$$

where E_{sm}, the energy penalty paid each time node v switches mode, is

$$\begin{aligned} E_{sm}(v) &= P_{ac2slm}(v) \cdot t_{ac2slm}(v) + P_{sl2acm}(v) \cdot t_{sl2acm}(v) \\ &\quad - P_{slm}(v) \cdot t_{sl2acm}(v) - P_{slm}(v) \cdot t_{ac2slm}(v) \end{aligned}$$

We assume that P_{acm},P_{slm},P_{ac2slm},P_{sl2acm},t_{ac2slm},t_{sl2acm} are given for all nodes and we can compute E_{sm} from these values.

Note that P_{ac2slm} and P_{sl2acm} account for both the dynamic and static power. Moreover, we consider E_{sm} constant, whenever Δt is large enough so that $\Delta t \cdot \Delta P > E_{sm}$. If any state registers are present in a process, they are not put in sleep mode, so that the state of the process can be preserved.

While each stage is defined by its ability to be executed in parallel with other stages, each process is defined by its ability to change mode independently of other processes[1].

2.3 Communication Channels

Communication channels are represented by directed edges, which connect processes or pipeline stages. Each edge can be implemented as a FIFO buffer. The amount of storage required for the buffer is given by the maximum number of tokens $b(e)$ at any time on the edge, which is determined by the schedule of the SDF. Since we do not modify the schedule inside a pipeline stage, we focus on the energy consumption of cross edges only.

The energy consumed on a cross edge is an increasing non-linear function of $b(e)$ and can be different for each edge, since the size of the tokens, the interconnect, and access patterns may be different.

We use the symbol $E_p(e, b(e))$ for the static and dynamic energy consumed on the memory implementing the channel e if e requires memory space for $b(e)$ tokens.

2.4 Scheduling and Throughput

A complete cycle or iteration of the graph consists of the execution of each stage s q_s times, where q_s is the corresponding entry for s in the repetitions vector. We say that there are q_s invocations or instances of s in one complete cycle of G. We denote s^i the ith instance of a stage s. Since G runs for an infinite number of times, $i \in Z^+$. For completeness we include instance s^0, which is not executed. Instance s^0 is considered to be completed before any other stage starts its first instance.

[1] Note that at this level each process represents a hardware unit. Since the graph G_s can have cycles and because of the definition of $l(v)$, our model does not prevent resource sharing.

Static scheduling imposes an ordering on the execution of events. A parallel schedule is a partial order on the set of the events. The partial order can be defined by a reflexive, anti-symmetric, and transitive relation R of precedence on the events. We denote as $\alpha \preceq \beta$ or $(\alpha,\beta) \in R$ the fact that event α happens before β happens. If α and β are not ordered by the relation, $(\alpha,\beta) \notin R$ and $(\beta,\alpha) \notin R$, the two events can occur in any order, even at the same time. An event can be the starting time or the ending time of the execution of a node. We can extend this relation to the execution of instances of stages. More specifically, we denote as $\alpha^i \prec \beta^j$ the fact that the ending time of instance i of stage α happens before the starting time of instance j of stage β. The relation $\prec$ is also transitive.

The edges of the graph define precedence constraints that restrict the number of available schedules that can be generated. Since all nodes (processes and stages) may carry state from one iteration to the next $\forall k \in 1..q_v : v^{k-1} \prec v^k$.

The buffer size of a channel should be large enough to store the maximum number of tokens present at that channel at any time. Suppose that $\prec$ defines a consistent and admissible schedule, then:

$$\forall (u,v) \in E, \forall i \in \mathbb{N}, \text{ let } j_{\max} \triangleq \max\{j \mid j \in \mathbb{Z}_0^+ \wedge v^j \prec u^i\}, \text{ then}$$

$$b(u,v) = \max_{\forall i}(i \cdot p(u,v) - j_{\max} \cdot c(u,v)) + w(u,v) \qquad (2)$$

The above formula holds because during the ith instance of the producer $(i-1)\cdot p(u,v)$ tokens have already been produced and $p(u,v)$ are being produced during that iteration. Meanwhile, $j_{\max}$ instances of the consumer have already completed execution and, therefore, $j_{\max} \cdot c(u,v)$ tokens have been consumed. To the total number of tokens present we need to add the $w(u,v)$ initial tokens.

We assume that the token production at the inputs is periodic. That means that if a complete cycle is executed within L_{cc}, then for each input edge i the period is $L_i = \frac{L_{cc}}{q_i}$, where q_i is the number of instances in a complete cycle. Note that, since the input graph is assumed to be consistent, we do not need to worry about the existence of the q_i values. Each stage should have an average invocation period of $L_s = \frac{L_{cc}}{q_s}$. Therefore, $l(G_s) \leq L_s$.

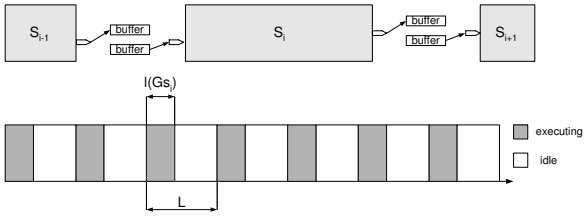

Figure 2: Execution of stage s_i with $x_i = 1$

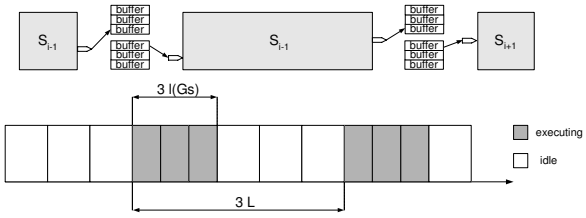

Figure 3: Execution of stage s_i with $x_i = 3$

3 Problem Formulation

As we saw in Equation 1, energy can be saved by switching the operation mode of some processes when enough idle time is available. One way to increase the energy savings could be to consecutively execute the stage for an integer number of times $x > 1$ and then allow its processes to be in sleep mode for a longer interval. This may increase the buffer requirements for the input and output channels. However, the switching mode penalty can now be shared across different instances for some of the processes of the stage.

The transformation can be described as replacing stage s by s', whose firing rules can be derived by multiplying by x the number of required inputs tokens for s. Moreover, s and s' have the same process graph, which for s' is repeated x times for each invocation, and, therefore, $l(G_{s'}) = x \cdot l(G_s)$. For the edges connected to s $\forall (t,s) \in E : c(t,s') = x \cdot c(t,s)$ and $\forall (s,t) \in E : p(s',t) = x \cdot p(s,t)$.

It is easy to prove that the topology matrix of the graph G after the transformation has the same rank and since the graph is acyclic, the graph is still consistent [1].

An example is shown in Figures 2 and 3. In the first figure for stage s_i the x value equals 1. Every L cycles s_i is idle for $L - l(G_{s_i})$ cycles. If this interval is long enough, some of the processes of s_i can be switched to sleep mode. The penalty for switching from active to sleep and back to active is E_{sm} every L cycles for those processes. In Figure 3 the addition of 2 extra buffers allows s_i to execute for three consecutive times. The idle time increases and potentially more processes can be shut down. Besides that, the penalty for the mode switch E_{sm} is paid once every $3 \cdot L$ cycles for each process. If there is a change in the input rate and the slack becomes zero, the two additional buffers can be shut down and the stage can operate as in the first case. We assume that such changes happen with a very low frequency, e.g. the changes are caused by the user of the system, and there is a small set of predefined values for the input rate. For each of these values we solve the following problem.

Given a multirate, consistent, hierarchical graph $G = (V,E)$ with the first level of hierarchy being a chain-structured multigraph, and a quality metric ρ, find the number of consecutive executions $x_s \in \mathbb{N}$ for each stage s, so that the energy savings are not less than $(1-\rho)\cdot E_{max}$, where E_{max} are the maximum energy savings that can theoretically be achieved by any solution to this problem.

4 Theoretical Exploration

In this section we reduce the search space of the solution. The solution space of the problem is $\mathbb{N}^{|S|}$. Using properties of the problem, the quality metric, and the energy penalty on the additional buffers we find an upper bound on the x values, making the solution space finite. This upper bound affects the complexity of the proposed algorithm and its running-time as shown in Sections 5 and 6.

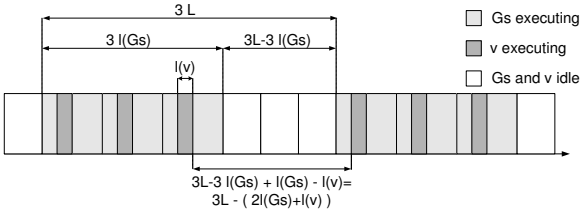

Figure 4: Idle time for type-1 processes

4.1 Energy Savings on Processes

Using Equation (1) we can explore the energy savings that can be obtained by any process. Suppose that G_s is the graph representing pipeline stage s and x_s is the number of consecutive executions of G_s. We can distinguish two types of processes.

Type-1 Processes Processes $v \in G_s$ for which

$$(l(G_s) - l(v)) \cdot \Delta P(v) < E_{sm}(v) \qquad (3)$$

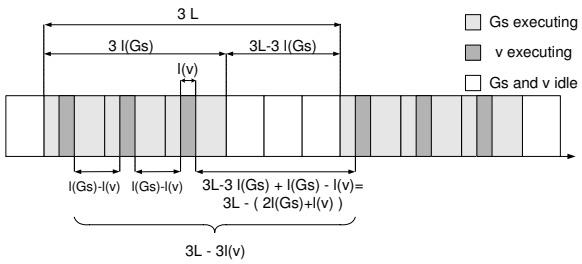

Figure 5: Idle time for type-2 processes

are Type-1 processes.

Process v is invoked x_s times with a period of $l(G_s)$ cycles (Figure 4). Let $st(v)$ and $et(v)$, $st(G_s)$ and $et(G_s)$ be the start and end time of intervals $l(v)$, $l(G_s)$, respectively. In the first iteration v can be invoked after $st(v)-st(G_s)$ cycles and in the x_sth iteration it can be put to sleep mode for $et(G_s)-et(v)$. Therefore, in the first and last iterations v is in active mode for $l(G_s)+l(v)$. In the rest x_s-2 iterations v is not switched to sleep mode, since Inequality (3) suggests that this would cause an energy loss. Therefore, the total time spent in active mode after $x_s \cdot L_s$ cycles is $(x_s-2)\cdot l(G_s)+l(G_s)+l(v)$ or $(x_s-1)\cdot l(G_s)+l(v)$ and the energy savings in this case are:

$$E_s(v) = (x_s \cdot L_s - (x_s-1)\cdot l(G_s) - l(v))\Delta \mathrm{P(v)} - E_{sm}(v)$$

Therefore, on average the energy savings in L_s cycles are:

$$E_s(v) = (L_s - \frac{(x_s-1)l(G_s)+l(v)}{x_s})\Delta \mathrm{P(v)} - \frac{E_{sm}(v)}{\mathrm{x_s}} \qquad (4)$$

Lemma 1 *The energy savings after L_s cycles of Type-1 process $v \in G_s$ are upper bounded by $(L_s - l(G_s))\cdot \Delta \mathrm{P(v)}$.*

Proof: : Follows from Equation 4 for $x \to \infty$. □

From Equation (4) we can express the energy savings difference obtained by increasing x_s from x_1 to x_2 as

$$\Delta E_s(v)(x_2,x_1) = \frac{x_2-x_1}{x_2 \cdot x_1}\Big(E_{sm}(v) - \Delta \mathrm{P(v)} \cdot (\mathrm{l(G_s)} - \mathrm{l(v)})\Big) \qquad (5)$$

which is always greater than zero, since $x_2 > x_1$ and because of Inequality (3). Since $\Delta E_s(v)(x_2,x_1)$ is positive, $E_s(v)$ is an increasing function of x_s for all Type-1 processes.

Type-2 Processes Processes $v \in G_s$ for which

$$(l(G_s)-l(v))\cdot \Delta P(v) \geq E_{sm}(v) \qquad (6)$$

are classified as Type-2 processes.

In this case during each of the $x-1$ executions of the pipeline stage G_s, the process can be put to idle mode for $l(G_s)-l(v)$ cycles.

Lemma 2 *The energy savings of a Type-2 process $v \in G_s$ are independent of x_s.*

Proof: Let x_s be the number of consecutive executions of G_s. Then the energy savings for process v in $x_s \cdot L_s$ cycles are:

$$\begin{aligned} E_s(v) &= \underbrace{(x_s \cdot L_s - (x_s-1)\cdot l(G_s) - l(v))\cdot \Delta \mathrm{P(v)} - E_{sm}(v)}_{\text{savings due to idle time after the } x_s\text{th ex. of } G_s} + \\ &\quad + \underbrace{(x_s-1)\cdot\Big((l(G_s)-l(v))\cdot \Delta \mathrm{P(v)} - E_{sm}(v)\Big)}_{\text{savings after each of the first } x_s-1 \text{ ex. of } G_s} \\ &= (x_s \cdot L_s - x_s \cdot l(v))\cdot \Delta \mathrm{P(v)} - \mathrm{x_s} \cdot E_{sm}(v) \end{aligned}$$

which means that in L_s cycles the energy savings are

$$E_s(v) = (L_s - l(v))\cdot \Delta \mathrm{P(v)} - E_{sm}(v)$$

Therefore, the energy savings are independent of x_s. □

For both Type-1 and Type-2 processes we need to multiply the above findings for L_s by q_s to obtain the energy savings in L_{cc} cycles.

4.2 Energy Penalty on Edges

In Section 2.3 we saw that the energy penalty on an edge is a non-linear, increasing function $E(e,b(e))$ with respect to the buffer size $b(e)$. Therefore, it is important to study how the buffer size of a cross edge is affected by the transformation, in order to estimate the energy penalty.

Determining the minimum buffer sizes for a sequential deadlock free schedule has been done in the past [6]. However, in our case we want to find the buffer sizes for a given parallel schedule, for which we are allowed to make as few assumptions as possible. For that reason we use formula (2).

A simplistic approach would be to consider the buffer size of a cross edge to be an increasing function of the x values of the stages. Even though this approach is simplistic it helps us draw some useful conclusions for the more general cases.

Unirate Case If the input graph is a unirate graph, a more realistic approach would be to consider the buffer size as the lcm function of the x values of the adjacent stages. In the graph before the transformation is applied $\tilde{q} = [11...1]$ and each stage is invoked once every L cycles. After the transformation the average rate of invocation for each instance remains the same. In $lcm(x_i,x_{i+1})\cdot L$ cycles we know that s_i is executed $\frac{lcm(x_i,x_{i+1})}{x_i}$ times, which correspond to $lcm(x_i,x_{i+1})$ instances before the transformation. Moreover, s_{i+1} is invoked $\frac{lcm(x_i,x_{i+1})}{x_{i+1}}$ times during the $lcm(x_i,x_{i+1})\cdot L$ cycles. Therefore, if $s_{i+1}^k \prec s_i^l$, then $\exists d_1 \in N$ such that $\forall l$

$$k = (\lceil \frac{l}{\frac{lcm(x_i,x_{i+1})}{x_i}} \rceil - d_1)\cdot \frac{lcm(x_i,x_{i+1})}{x_{i+1}}$$

Because of (2), $b(e) = \max_{\forall l}(l\cdot p(e) - k\cdot c(e)) + w(e)$. For $l = d_2 \cdot \frac{lcm(x_i,x_{i+1})}{x_i}$, where $d_2 \in \mathbb{N}$, $b(e)$ is maximized:

$$b(e) = d_2 \cdot \frac{lcm(x_i,x_{i+1})}{x_i}\cdot p(e) - (d_2-d_1)\cdot \frac{lcm(x_i,x_{i+1})}{x_{i+1}} c(e) + w(e)$$

Since after the transformation $p(e) = x_i$ and $c(e) = x_{i+1}$,

$$b(e) = d_2 \cdot \frac{lcm(x_i,x_{i+1})}{\not{x}_i}\cdot \not{x}_i - (d_2-d_1)\cdot \frac{lcm(x_i,x_{i+1})}{\not{x}_{i+1}}\cdot \not{x}_{i+1} + w(e)$$

$$\Rightarrow b(e) = d_1 \cdot lcm(x_i,x_{i+1}) + w(e)$$

In this case the buffer size and, because of that, the energy penalty are increasing functions with respect to the $lcm(x_i,x_{i+1})$.

Multirate Case The multirate case is similar to the unirate case except that the entries in the repetition vector need to be taken into account as well. Without the transformation, stage s_i completes q_i executions and stage s_{i+1} completes q_{i+1} executions during one complete cycle. After the transformation that is not necessarily true. However, for the transformed graph we know that $lcm(x_i,x_{i+1},q_i,q_{i+1})$ defines a period during which s_i completes $\frac{lcm(x_i,x_{i+1},q_i,q_{i+1})}{x_i}$ and s_{i+1} $\frac{lcm(x_i,x_{i+1},q_i,q_{i+1})}{x_{i+1}}$ executions. Solving as for the unirate case, we can find that the buffer sizes, $b(e) = d_1 \cdot lcm(x_i,x_{i+1},q_i,q_{i+1}) + w(e)$ are an increasing function of $lcm(x_i,x_{i+1},q_i,q_{i+1})$.

Since in all the above cases the information that we have about each edge is that they are increasing functions of $b(e)$, we can collapse all edges between two stages to one. The new function is given by: $E_p^{(i,i+1)} = \sum_{\forall e_{\text{connecting i to i+1}}} E_p(e,b(e))$. Function $E_p^{(i,i+1)}$ is also an increasing function with respect to the buffer sizes on all channels between stages i and $i+1$.

4.3 Energy Savings Limit and x_{max}

From the previous sections we can derive the formula for the total energy savings $E_t = \sum_{\forall s}\sum_{\forall v \in G_s} E_s(v) - \sum_{i=0}^{|S|} Ep^{(i,i+1)}$. This is the function we want to maximize. In this section we derive a bound on the x values of the stages to prune the search space of the problem. We start again from the simplistic case assuming that the buffer sizes are an increasing function of the x_is and move to more realistic cases.

We denote as $b(i)$ the buffer sizes of edges that connect stages i and $i+1$.

Let $v \in G_s$ be a Type-1 process and

$$C(v) \triangleq \frac{L \cdot \Delta P(v) - l(G_s) \cdot \Delta P(v)}{E_{sm}(v) - \Delta P(v) \cdot (l(G_s) - l(v))}$$

a constant for that node that depends only on the input graph. Let

$$C(G) \triangleq \min_{\forall v \in \text{Type-1}} C(v)$$

Then the following Lemma can be proved.

Lemma 3 *If G is a unirate graph and $b(i) = f(x_i, x_{i+1})$ are increasing functions with respect to both x_i and x_{i+1}, and $\tilde{x} = [x_1 x_2 ... x_{|S|}]$ is the optimal solution resulting in maximum total energy savings E_t^{max}, then for any $0 < \rho < 1$ and $x_{max} = \lceil \frac{1}{\rho \cdot C(G)} \rceil$, there exists $\tilde{x}' = [x'_1 x'_2 ... x'_{|S|}]$ with $\forall i \in 1..|S| : 1 \le x'_i \le x_{max}$, for which the total energy savings E'_t are greater or equal to $(1-\rho) \cdot E_t^{max}$.*

For example, if the designer chooses $\rho = 0.05$, we can find x_{max} from the input graph and ρ. Then Lemma 3 states that there exists $\tilde{x}'$, whose entries are all less or equal to x_{max} and the energy savings for $\tilde{x}'$ are $E'_t \ge 0.95 \cdot E_t^{max}$.

Unirate Graphs A similar approach can be followed for $b(i) = f_i(lcm(x_i, x_{i+1}))$, where for all $i \in 1..|S|$, $f_i : \mathbb{N} \to \mathbb{N}$ is an increasing function.

Lemma 4 *If G is a unirate graph, $f_i : \mathbb{N} \to \mathbb{N}$ is an increasing function, $b(i) = f_i(lcm(x_i, x_{i+1}))$ for each cross edge $(i, i+1)$, and $\tilde{x} = [x_1 x_2 ... x_{|S|}]$ is the optimal solution resulting in maximum total energy savings E_t^{max}, then for any $0 < \rho < 1$ and $x_{max} = (\lceil \frac{1}{\rho \cdot C(G)} \rceil)^2$, there exists $\tilde{x}' = [x'_1 x'_2 ... x'_{|S|}]$ with $\forall i \in 1..|S| : 1 \le x'_i \le x_{max}$, for which the total energy savings are greater or equal to $(1-\rho) \cdot E_t^{max}$.*

Multirate Graphs For multirate graphs the buffer sizes depend on the q values. Using a similar approach as for unirate graphs could result in describing x_{max} as a function of q. The q values though can grow exponentially with the input graph [2] and, therefore, a more general method is needed to derive x_{max}.

From Lemma 1 we can derive a bound on the energy savings that can be achieved. Let $E_s(\infty)$ be the sum of the savings for Type-1 processes when x goes to infinity, and $E_s(1)$ when $x = 1$. Also let $E_p(1)$ be the value of the energy penalty when $x = 1$. Let y_i^{max} be the minimum value, for which the energy penalty becomes $E_p^{(i,i+1)}(e, y_i^{max}) \ge E_s(\infty) - E_s(1) + E_p(1)$. We know that increasing y_i to a value greater than y_i^{max} can cause only energy loss, since the savings cannot become greater than $E_s(\infty)$ and $E_p^{(i,i+1)}(e, y_i)$ is increasing with respect to y_i. Therefore, any $y_i > y_i^{max}$ causes an energy penalty that exceeds any energy savings obtained by the Type-1 processes.

Since the energy penalty for all edges is already given (most probably in form of an array of values), binary search can be applied to each of the $(|S|+1)$ functions $E_p^{(i,i+1)}$ to find y_i^{max}. The binary search procedure can start with a very large value Y as the maximum value for y that is determined by computational precision limits or area constraints. We know that $y_i = lcm(x_i, x_{i+1}, q_i, q_{i+1})$. Since we also have ${x_{max}}^i \le lcm({x_{max}}^i, x_{i+1}, q_i, q_{i+1}) = y_i$ and ${x_{max}}^i \le lcm(x_{i-1}, {x_{max}}^i, q_{i-1}, q_i) = y_{i-1}$, it holds ${x_{max}}^i \le \min(y_{i-1}, y_i)$. If for each stage i ${x_{max}}^i = \min(y_{i-1}, y_i)$, then $x_{max} = \max_{\forall i}({x_{max}}^i)$ can be chosen as the maximum value for the whole design. Any increase of x above that value for any of the stages causes energy loss compared to the case, in which all x values are 1.

Lemma 5 *For the optimal solution $\tilde{x}$ of the multirate problem the following property holds: $\forall i \in 1..|S| : 1 \le x_i \le x_{max}$.*

This method can be applied to a unirate graph as well, and, therefore, we use it in conjunction with the approaches for unirate graphs described above. We use the minimum of the two x_{max} values produced. The running time of the binary search method described above is $O(|S| \cdot \log Y)$.

5 Dynamic Programming Solution

In this section we describe a dynamic programming algorithm which can determine the x values for maximum energy savings given a quality metric. The algorithm is needed because the size of the solution space is still large after bounding the x values with x_{max}. Exhaustive search requires $O({x_{max}}^{|S|})$ steps to find the x values for maximum energy savings.

In Figure 6 the algorithm can be seen. The inputs are the graph which is partitioned in pipeline stages and a quality metric in case the graph is unirate. After initialization, the algorithm determines x_{max} using the procedures described in Section 4. The purpose of the rest of the algorithm is to solve independently the problem for each subchain and combine the solutions to find the optimal solution for the chain-structured graph.

The intuition behind the DP solution is that the values x_{i-1} and x_{j+1} are the only external values that can affect the optimal solution for a subchain from stage i to stage j. More specifically, if $i, ..., j$ is a subchain with $1 \le i < j \le |S|$, the best configuration for this subchain, i.e. the vector of x values $[x_i, ..., x_j]$ that provides maximum energy savings, depends only on the x values of the stage exactly before the subchain, i.e., x_{i-1}, and the stage after the subchain, i.e., x_{j+1}, (Figure 7). Therefore, an ${x_{max}}^2$ matrix can be constructed storing the maximum energy savings that can be obtained for that subchain for each value of the pair (x_{i-1}, x_{j+1}). Such a matrix can gradually be built for all possible subchains of the problem. This array is denoted as $e_s[|S|][|S|][x_{max}][x_{max}]$ in the algorithm of Figure 6. As an example, element $e_s[i, j, x_{i-1}, x_{j+1}]$ holds the best configuration for subchain starting at stage i and ending at stage j, when the x value for stage $i-1$ is x_{i-1} and for stage $j+1$ it is x_{j+1}.

After finding x_{max} the algorithm starts by creating the e_s array for subchains of length 0. The entries filled during this phase are the ones on the main diagonal of the simplified array e_s of Figure 7. For each $e_s[i][i]$ the ${x_{max}}^2$ matrix is built from the energy savings for stage i and the energy penalty of both cross edges $(i-1, i), (i, i+1)$ for that stage. In the second phase the algorithm fills the entries for subchains with two elements. Finally, in the third phase the energy savings for all remaining subchains are found. The reason for the separate treatment of subchains with two and more than two elements is to make sure that the energy penalty for the same cross edge is not taken twice into account. The maximum energy savings for the whole graph are stored at position $e_s[1][|S|][1][1]$. This entry

Algorithm DP-for x values
Input: A chain structured SDF graph $G=(S,E)$ representing the pipeline stages, a quality metric ρ which will be used if G is unirate, and functions $E_p^{(i,i+1)}(x_i,x_{i+1})$ returning the energy overhead for cross edges between stages i and i+1.
Output: Two arrays $x_{best}[|S|,|S|,x_{max},x_{max}]$ and $e_s[|S|,|S|,x_{max},x_{max}]$ from which the optimal solution can be extracted.

```
InitEs(G);
x_max ← DetermineXmax(G,ρ);
for i ← 1 to |S| do // main diagonal d = 0
  for x_{i-1} ← 1 to x_max do
    for x_{i+1} ← 1 to x_max do
      for x_i ← 1 to x_max do
        e_s^new ← E_s^i(x_i) − E_p^{(i-1,i)}(x_{i-1},x_i) − E_p^{(i,i+1)}(x_i,x_{i+1})
        if (e_s[i,i,x_{i-1},x_{i+1}] < e_s^new) then
          e_s[i,i,x_{i-1},x_{i+1}] ← e_s^new
          x_best[i,i,x_{i-1},x_{i+1}] ← x_i
for i ← 1 to |S|−1 do // init step for d = 1
  for x_{i-1} ← 1 to x_max do
    for x_{i+2} ← 1 to x_max do
      for x_node ← 1 to x_max do
        // x_node represents x_i and x_{i+1} in this loop
        e_s^new1 ← e_s[i,i,x_{i-1},x_node] + E_s^{i+1}(x_node)
                   − E_p^{(i+1,i+2)}(x_node,x_{i+2})
        e_s^new2 ← e_s[i+1,i+1,x_node,x_{i+2}] + E_s^i(x_node)
                   − E_p^{(i-1,i)}(x_{i-1},x_node)
        e_s^new ← max(e_s^new1, e_s^new2)
        node ← (e_s^new1 > e_s^new2)
        if (e_s[i,i+1,x_{i-1},x_{i+2}] < e_s^new) then
          e_s[i,i+1,x_{i-1},x_{i+2}] ← e_s^new
          x_best[i,i+1,x_{i-1},x_{i+2}] ← (node,x_node)
for d ← 2 to |S|−1 do //diagonal count
  for i ← 1 to |S|−d do
    j ← i+d
    for k ← 1 to j−i−1 do
      for x_{i-1} ← 1 to x_max do
        for x_{j+1} ← 1 to x_max do
          for x_{i+k} ← 1 to x_max do
            e_s^new ← e_s[i,i+k−1,x_{i-1},x_{i+k}] + E_s^{i+k}(x_{i+k}) +
                      + e_s[i+k+1,j,x_{i+k},x_{j+1}]
            if (e_s[i,j,x_{i-1},x_{j+1}] < e_s^new) then
              e_s[i,j,x_{i-1},x_{j+1}] ← e_s^new
              x_best[i,j,x_{i-1},x_{j+1}] ← (i+k,x_{i+k})
Return x_best, e_s;
```

Figure 6: Pseudocode for the dynamic programming algorithm

represents the whole chain with $x_0 = x_{|S|+1} = 1$. As mentioned before, we assume that stages s_0 and $s_{|S|+1}$ are external and we have no control over them. Therefore, their x values remain 1. Array $x_{best}[|S|,|S|,x_{max},x_{max}]$ stores the decision taken at each step and information necessary to retrieve the optimal solution.

The algorithm searches all possible values from 1 to x_{max} for x, at each subproblem and, therefore, it solves each subproblem optimally. Moreover, since the subproblems are independent, the algorithm finds the solution with the maximum total energy savings for all $1 \le x_i \le x_{max}$.

At each step the algorithm computes the energy savings and energy penalty using functions E_s and E_p. The function for the energy savings can be implemented as described in Section 4. More specifically, during initialization, i.e. InitEs(G) step, we can find the energy savings for the Type-2 processes, which are independent of x and, therefore, we do not need to recompute them during the iterations of

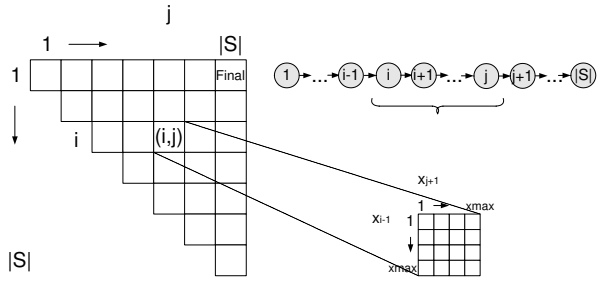

Figure 7: Dynamic Programming Algorithm. The solution for subchain (i,j) depends only on values x_{i-1} and x_{j+1}. In a ${x_{max}}^2$ array the best configuration of (i,j) is stored for each value combination of x_{i-1} and x_{j+1}.

the algorithm. For Type-1 processes of each stage s we can use the following formula to find the energy savings for a specific x

$$E_s(x) = \sum_{\forall v \in G_s} L_s \cdot \Delta P(v) - \frac{x-1}{x} \sum_{\forall v \in G_s} l(G_s) \cdot \Delta P(v) - \frac{1}{x} \sum_{\forall v \in G_s} (l(v) \cdot \Delta P(v) + E_{sm}(v))$$

It is clear from the equation above that all summations can be computed during the initialization step (InitEs). Then E_s can be computed in constant time for each new value of x. It is assumed that the functions E_p are given by the user in the form of an array and, therefore, the energy penalty for a pair of x values can be returned at constant time. Consequently, the algorithm's complexity is $O(|S|^3 \cdot {x_{max}}^3 + |S| \cdot \log Y)$ and its memory space requirements are $O(|S|^2 \cdot {x_{max}}^2)$.

Theorem 1 *The solution found by the dynamic programming algorithm produces total energy savings $E_t^{alg}(x_{max})$, which are at least $(1-\rho) \cdot E_t^{max}$, if the energy penalty at the cross edges $(i,i+1)$ is an increasing function of both x_i, x_{i+1} and x_{max} is given by $x_{max} = \lceil \frac{1}{\rho \cdot C(G)} \rceil$.*

Theorem 2 *The solution found by the dynamic programming algorithm produces total energy savings $E_t^{alg}(x_{max})$, which are at least $(1-\rho) \cdot E_t^{max}$, if the energy penalty at the cross edges $(i,i+1)$ is an increasing function of $lcm(x_i,x_{i+1})$ and x_{max} is given by $x_{max} = {x_g}^2$, $x_g \ge \lceil \frac{1}{\rho \cdot C(G)} \rceil$.*

Theorem 3 *The solution found by the dynamic programming algorithm produces total energy savings $E_t^{alg}(x_{max}) = E_t^{max}$, if the energy penalty at the cross edges $(i,i+1)$ is an increasing function of $lcm(x_i,x_{i+1},q_i,q_{i+1})$ and x_{max} is given by the binary search procedure described above for multirate graphs.*

6 Experimental Results

The algorithm was implemented as a C++ program taking consistent graphs as an input and determining the x values for each pipeline stage. For the experiments we normalized the power of all components using the static power of the 32-bit latch. The static power of 32-bit output multipliers was set to 25 and the 32-bit cla adders 4 times that of the latch. The static power of the decoding logic for the channels was considered at the same level as the static power of the latches. On the channels the dynamic power increase with x, caused by the extra wiring and control, was considered 50% of the static

Application	CD-to-DAT (multirate,#stages=3)			K-means (unirate, #stages=10)			K-means (unirate, #stages=3)		
Input Rate	50%	25%	12.5%	11.1 %	8.33%	6.67%	33.33 %	25%	12.5%
Alg. Exec. Time(sec)	2.97	2.97	2.98	144.39	29.26	3.37	5.79	0.7	0.06
x_{max}	71	71	71	169	100	49	100	49	16
Increase in En. Savings	15.17%	5.25%	2.27%	N/A	107.31%	6.71%	900.38%	24.25%	0%

Table 2: Experimental Results for several input rates. The input rates are expressed as a percentage of the worst case input rate. The increase in energy savings is "N/A" when the energy savings of power gating with x=1 for all stages are 0.

Application Input Rate	10-stage K-means 6.67%		3-stage K-means 25%	
ρ	0.90	0.95	0.90	0.95
Alg. Exec. Time(sec)	3.37	351	0.7	44.1
x_{max}	49	225	49	196
Increase in En. Savings	6.71%	6.71%	24.25%	24.25%

Table 3: Effect of the ρ value on running-time

power increase. The switching mode overhead was considered equal with the energy savings obtained by 10 cycle time slack.

We applied the algorithm on three pipelined architectures. The first is the CD to DAT sample-rate conversion graph adopted from [3]. Each of the 3 SDF actors of the multirate graph was considered a pipeline stage. The FIR filters were assumed to be 4-tap filters implemented with multipliers. Upsampling, filtering, and downsampling units were considered independent processes forming together one stage. So, in total there were 3 stages in the first level of hierarchy executing in parallel. The second application was the unirate 10-stage pipelined K-means clustering with euclidean distances adopted from [11]. The third was a 3-stage pipelined architecture for K-means clustering. In the latter case the first two, intermediate four, and last four pipeline stages of the 10-stage pipelined K-means were merged to form a 3-stage pipeline. Figure 8 shows the energy savings obtained by our algorithm compared to the energy savings taken by applying power gating only (all x values equal to 1). In all graphs a mode switch occurred for a process only if the energy savings obtained by the switch exceeded the energy overhead E_{sm}.

For each application we tried several input rates. As stated in the introduction we assume that the set of input rates is predefined and changes in the input rates happen with a low frequency (e.g. user-controlled). For higher input rates the idle time in each complete cycle is shorter. Therefore, the energy savings obtained by power gating (all x values equal to 1) are low or zero. For these cases applying the proposed technique has a significant impact as seen from the last row of Table 2. As the input rate is reduced, mode transitions occur less often. The energy consumption because of the mode switch overhead becomes less significant and, consequently, the additional savings obtained by the proposed technique decrease.

For the 3-stage pipelined K-means gains are produced in higher input rates than for the 10-stage pipelined architecture. The reasons for this are that more Type-1 processes are sharing the penalty paid on the cross edges of one stage, and that the slack for each process is increased because the latency of each stage ($l(G)$) is longer. In Table 2 the results are shown. Finally, in Table 3 the effect of the ρ value on running-time can be seen. In this case the energy savings are the same for different ρ values. However, a higher ρ value offers a guarantee for the proximity to the optimal solution, whereas a lower ρ value results in a shorter running-time.

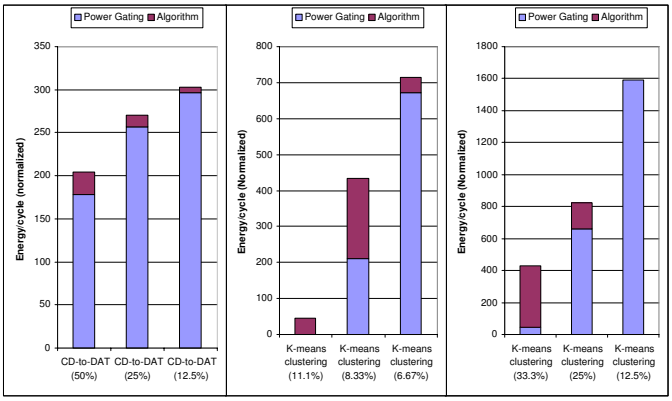

Figure 8: Energy savings obtained by our technique and power gating (red and blue) compared to the savings obtained by applying only power gating (blue) for several input rates of 3 applications: CD-to-DAT (left), 10-stage pipeline K-means (middle), 3-stage pipeline K-means (right). The input rates are expressed as a percentage of the worst case input rate below the name of the application.

7 Conclusion

In this paper we presented an approach to reduce energy consumption using power gating. An analysis framework was presented and a theoretical bound on the number of consecutive iterations was derived for chain-structured pipelines. An algorithm was developed that can give an optimal solution for the total energy savings. In the future we plan to work on evaluating scheduling techniques that can reduce the energy overhead of DVS and ABB.

References

[1] E. A. Lee, D. G. Messerschmitt; "Static Scheduling of Synchronous Data Flow Graphs"; IEEE Transactions on Computers, Jan 1987

[2] J. L. Pino, S. S. Bhattacharayya, E. A. Lee; "A Hierarchical Multiprocessor Scheduling Framework for Synchronous Dataflow Graphs";UCB/ERL M95/36, May 30, 1995

[3] P. K. Murthy, et. al.; "Minimizing Memory Requirements for Chain-Structured Synchronous Dataflow Programs"; ICASSP 94

[4] C. Im, H. Kim, S. Ha; "Dynamic Voltage Scheduling Technique for Low-Power Multimedia Applications Using Buffers"; ISLPED 2001

[5] K.S. Khouri, N. K. Jha; "Leakage Power Analysis and Reduction During Behavioral Synthesis"; IEEE TVLSI Vol. 10, No 6, December 2002

[6] M. Ade, et. al.; "Data Memory Minimization for Synchronous Data Flow Graphs Emulated on DSP FPGA Targets"; DAC 1997

[7] S. Ritz, et. al.;"Optimum Vectorization of Scalable Synchronous Data Flow Graphs"; Application-Specific Array Processors 1993

[8] Z. Hu, A. Buyuktosunoglu; "Microarchitectural Techniques for Power Gating of Execution Units"; ISLPED 2004

[9] T. D. Burd, R. W. Brodersen; "Design Issues for Dynamic Voltage Scaling"; ISLPED 2000

[10] C. H. Kim, K. Roy; "Dynamic V_{TH} Scaling Scheme for Active Leakage Power Reduction"; DATE 2002

[11] M. Estlick, M. Leeser, et. al.; "Algorithmic Transformations in the Implementation of K-means Clustering on Reconfigurable Hardware"; FPGA 2001

[12] N. Liveris, H. Zhou, P. Banerjee; "A Dynamic-Programming Algorithm for Reducing the Energy Consumption of Pipelined System-Level Streaming Applications"; TR-NWU-EECS-07-09, 2007 (http://www.eecs.northwestern.edu/research/tech_reports/)