Multi-module memory has been employed in high-end digital signal processing system (DSP). It provides high memory bandwidth and low power operating mode for energy savings. However, making full use of these architectural features is a challenging problem for code optimization. In this paper, we propose an integer linear programming (ILP) model to optimize the performance and energy consumption of multi-module memories by solving variable assignment, instruction scheduling and operating mode setting problems simultaneously. The combined effect of performance and energy saving requirements has been considered as well. Specially, we develop two optimization techniques to improve the computation efficiency of our ILP model. The experimental results show that the optimal performance and energy solution can be achieved within a reasonable amount of time.

1. Introduction

Programmable Digital Signal Processor (DSP) is the most widely used processor for high-end consumer electronic products. In order to meet ever-growing performance requirements, many DSP cores are equipped with multiple data memory modules, such as Motorola DSP563xx [1] and NEC μPD771xx DSP, etc. Multi-module memory supports higher memory bandwidth than monolithic memory by permitting multiple data memory accesses to occur in parallel when the referenced variables belong to different data memory modules. However, performance gain derived from using multi-module memories strongly depends on the architecture-specific code optimization pass in the compiler. Variable assignment and instruction scheduling are two critical problems that need to be addressed during code generation [2–4].

In a consumer electronic device, embedded memory is a major energy consumer [5], especially for the data-intensive application, e.g. digital signal processing. Energy optimization considered in this work takes advantage of the low power operating mode of memory modules. The effectiveness of the low power operating mode is mainly determined by the time interval between successive accesses to the same memory module. That further complicates the problems of variable assignment and instruction scheduling. Moreover, in real-time applications, stringent timing constraint is prerequisite for energy optimization.

Many works have been done on pure performance-oriented [6–13] or pure energy-oriented [14–16,5,17–22] optimization, but the effects of combing both performance and energy requirements are seldom considered. In the conventional method, variables are partitioned to different memory modules based on a graph model before generating instruction schedule with a heuristic scheduling algorithm. The main distinction between these methods lies on their graph model, such as the IG [6] and VIG [7] models. In addition, Sudarsanam and Malik [8] presented a specialized optimization for Motorola DSP56000 based on simulated annealing. Recently, optimizations that reduce the energy consumption of multi-module memories have received more attention. In [14,15], some techniques to optimize the data layout for energy savings are proposed. Wang and Hu [16] proposed a VPIS algorithm to balance energy saving and performance requirements. To the best of our knowledge, there is no ILP model proposed for energy and performance optimization of multi-module memory. ILP-based scheduling techniques are only used for pure performance oriented optimization [23–26].

In this paper, we propose an ILP-based technique named Performance and Energy Optimal Scheduling (PEOS) to generate performance and/or energy optimal instruction schedule for multi-module memories in DSPs. The experimental results show that the optimal schedule with respect to performance and/or energy saving can be achieved with our ILP model in a reasonable amount of time. The main contributions of our work are:
We formulate the variable assignment, instruction scheduling and operation mode setting problems in a unified ILP framework. The limited register constraint is also considered. Simultaneously addressing these interdependent problems can guarantee a global optimal solution.

- Based on this unified ILP model, we can achieve multiple optimization objectives including: minimal schedule length, minimal energy consumption, and minimal energy consumption within a timing constraint. In some cases, we can achieve a solution that is optimal for both time and energy consumption.

- We model the nonlinear conditions “c = 0 if and only if r = 0/1” with linear constraints.

- We develop two methods to improve the execution efficiency of our ILP model. Our experimental results show that the proposed approximation method can achieve 95% of the energy savings on average with about 10% of computation time of the pure ILP method.

The rest of this paper is organized as follows. Section 2 introduces some basic models. A motivational example and formal problem statements are given in Section 3. The ILP model and related execution efficiency optimization techniques are proposed in Sections 4 and 5. Experimental results and conclusion are provided in Sections 6 and 7.

2. Basic models

2.1. Architectural model

In this work, we focus on an architecture in which the memory subsystem is composed of multiple data memory modules with a low power operating mode. The architectural units of interest are shown as the block diagram in Fig. 1. The data memories are single-ported SRAM which can be only accessed by the Memory-access Unit (MU) coupled with them. Each functional unit (ALU, MU) has single clock-cycle latency, which is a common feature among most DSP implementations. The Motorola DSP563xx and NEC \( \mu P D 771 x x \) DSP processors are real implementations with similar architecture. The main advantage of this architecture is that multiple data access operations can be executed in parallel.

Energy model. We only consider the energy consumption of data memory modules. We assume that each data memory module has two operating modes: active (full-power) and standby (low-power) mode. A memory module can service a memory access request only when it is in the active mode. When a memory module enters the standby mode, the data in SRAM can be retained by a very low standby current. The total energy consumption of SRAM-based data memory modules can be modeled, at high level abstraction, as summation of dynamic and static energy consumption. The dynamic energy consumption \( E_{\text{dynamic}} \) is modeled as follows.

\[
E_{\text{dynamic}} = N_{\text{access}} \times E_{\text{access}}
\]

where \( N_{\text{access}} \) denotes the number of memory accesses, \( E_{\text{access}} \) represents the required energy per memory access. In this paper, we assume that \( E_{\text{access}} \) is 0.29 nJ [27]. The static energy consumption \( E_{\text{static}} \) depends on the static energy model of the SRAM shown in Fig. 2 [28]. The energy consumption of reactivating a memory module from standby mode is assumed to be equal to energy consumption of active mode. Therefore, \( E_{\text{static}} \) can be stated as

\[
E_{\text{static}} = E_a \times N_m \times SL - (E_a - E_s) \times C_s
\]

where \( N_m \) is number of data memory modules, \( SL \) is the schedule length of given application, \( C_s \) is the total number of cycles in standby mode, and \( E_a \) and \( E_s \) are the energy consumptions per cycle of active and standby modes, respectively. The similar static energy model can be found in Samsung’s sync pipelined burst SRAM [29].

The operating mode transition of each data memory module is independently controlled by software via the memory controller. Only when the consecutive idle time is enough long to compensate for the transition time, the memory can be switched into standby mode.

2.2. MDFG model

In this paper, we use Memory-access Data Flow Graph (MDFG) as the description of a given program.

**Definition 1.** A MDFG \( G = (V, E, t, \text{var}) \) is a directed graph, where \( V \) is set of operation nodes, \( E \) is the edge set that defines the precedence relationships over the nodes in \( V \), \( t(v) \in [A, M] \) represents the type of a node \( v \in V \), \( \text{var}(v) \) represents the variable accessed by a node \( v \in \{t(v) = M\} \).

An example of MDFG is shown in Fig. 3. Arithmetic/logic operations \( (t(v) = A) \) are denoted by circles, and memory accesses \( (t(v) = M) \) are denoted by rectangles. For the ease of explanation, we assign each node a number. In our architectural model, all operations are single cycle operations, so we omit the definition of execution latency of each operation. Note that the name of array in MDFG is not the same as array in the source code. In order to simplify spreading array elements, we can divide the large size array into some small size array and give them a new name. In this paper, we treat arrays in MDFG is indivisible.

3. Motivational example and problem statement

3.1. Motivational example

In this paper, the performance of a given schedule is measured by its schedule length. Fig. 3 shows the MDFG for the motivational example. Fig. 4 illustrates that different schedules, as well as variable assignments, can affect the schedule length and energy consumption.
Fig. 4a shows the result generated by the VPIS method [16], which partitions the variables to different memory modules by the MAG graph model then generates, with list scheduling, a schedule based on the variable layout. The resulting schedule includes 12 control step (CS), which is not optimal with respect to schedule length because the critical path of the given MDFG is 11. The main reason is that variable partitioning in the VPIS method is energy aware, but might lead to performance degradation. VIG method [7] is similar to VPIS, but its variable partitioning is only performance aware. For this example, the schedule generated by VIG method has the optimal schedule length. Fig. 4c shows the results generated by our PEOS method, which can guarantee that the schedule length is optimal. From this comparison, we know that variable assignment and instruction scheduling are two interdependent problems for achieving the optimal schedule length.

The “M1” and “M2” columns in Fig. 4 show the memory accesses of the two memory modules. The operating mode transitions are clearly marked as well. Comparing the schedules in Fig. 4b and c, we find that the energy consumption of the schedule in Fig. 4c is 9.6% lower than the schedule in Fig. 4a, and the schedule length of Fig. 4c is 8.3% shorter than Fig. 4a. This comparison illustrates that energy saving and performance are not two fully conflicting requirements. It is possible to generate a schedule with the minimal schedule length and energy consumption at the same time with our ILP-based approach.

3.2. Problem statement

Given a MDFG $G$ and multi-module memory configuration, our goal is to minimize the schedule length and/or energy consumption of the resulting schedule. To achieve these goals, our proposed method will solve the following problems simultaneously with a unified ILP model:

- **Variable assignment.** Assigning variables to memory modules in a performance- and energy-aware fashion. We assume that array variables are indivisible.
- **Instruction scheduling.** Mapping operations on appropriate functional units and determining its start time. A legal schedule should obey the following constraints: (1) data dependencies, (2) limited register constraints, (3) type matching, i.e. operation types should match with functional unit types, (4) schedule deadline constraint for energy minimization.
- **Operating mode setting.** Determining the operating mode of each memory module at each control step.

4. The ILP model

In this section, we formulate our ILP model to solve all the problems raised in Section 3.2. We begin by establishing some notations shown in Table 1.

### 4.1. Performance optimization

A formal model of the instruction scheduling problem under resource constraints can be achieved by using binary decision variables $X = \{x_{ij,k}, \; i = 1, 2, \ldots, N; \; j = 1, 2, \ldots, S; \; k = 1, 2, \ldots, F\}$. A decision variable $x_{ij,k}$ is 1 only when operation $i$ is executed in them.

### Table 1

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>Number of nodes in the given MDFG $G$</td>
</tr>
<tr>
<td>$N_{\text{access}}$</td>
<td>Number of memory access operations in $G$</td>
</tr>
<tr>
<td>$N_V$</td>
<td>Number of variables in $G$</td>
</tr>
<tr>
<td>$\varepsilon_{ij}$</td>
<td>An edge $e(i,j)$ in $G$</td>
</tr>
<tr>
<td>$N_{MM}$</td>
<td>Number of data memory modules</td>
</tr>
<tr>
<td>$N_{\text{ALU}}$</td>
<td>Number of ALUs</td>
</tr>
<tr>
<td>$N_{\text{R}}$</td>
<td>Number of registers available for memory operations</td>
</tr>
<tr>
<td>$E_d$</td>
<td>Energy saving $E_d = E_a - E_s$</td>
</tr>
</tbody>
</table>
control step \( j \) on function unit \( k \). The number \( S \) represents an upper bound on schedule length, because the schedule length is unknown. The bound \( S \) can be given deadline or computed by a fast heuristic scheduling algorithm. All the function units are indexed from 1 to \( F \), where \( F = N_m + N_a \). A functional unit \( k \) is an ALU, if \( 1 \leq k \leq N_m \), otherwise it is a memory unit.

Operation mapping. Every operation is executed exactly once on a type matching functional unit.

\[
\sum_{j=1}^{S} \sum_{k \in \mathcal{T}(i)} x_{i,j,k} = 1, \quad \forall i \in \{1, 2, \ldots, N\} 
\]  

(3)

\( T(i) = \{k | t_{in}(k) = t(i), k \in \{1, F\} \} \) is the set of all functional units that match operation \( i \), where \( t_{in}(k) \) represents the type of functional unit \( k \), \( t(i) \) represents the type of operation \( i \) (refer to Definition 1).

We define a function \( p(j) \) to denote the number of operations executed in control step \( j \).

\[
p(j) = \sum_{i=1}^{N} \sum_{k \in \mathcal{T}(i)} x_{i,j,k}, \quad \forall j \in \{1, 2, \ldots, S\} 
\]  

(4)

\( p(j) \) is an integer value between 0 and \( F \). For example as the schedule table shown in Fig. 4a, \( p(1) = 2 \). If no operation is executed at step \( j \), then \( p(j) = 0 \). We define binary decision variables: \( Q = \{q_{ij} \mid j = 1, 2, \ldots, S\} \) based on \( p(j) \) to denote whether a control step is valid or not. The value of a decision variable \( q_{ij} \) is defined as

\[
q_{ij} = \begin{cases} 
0 & \text{if } p(j) = 0 \\
1 & \text{if } p(j) = 1, 2, \ldots, F 
\end{cases} 
\]  

(5)

If \( q_{ij} = 1 \) then step \( j \) is a valid step in the resulting schedule. Unfortunately, (5) is a nonlinear condition. We show that this type of nonlinear condition can be modeled with linear constraints via the following theorem.

**Theorem 1.** Let \( c \) be an integer variable whose value is between 0 and \( L \), and \( r \) be a binary variable. The nonlinear condition: “\( c = 0 \) if and only if \( r = 0 \)” can be modeled as the following linear form:

\[
r \leq c \leq L \times r 
\]  

(6)

**Proof.** Let us prove the following statements are correctly modeled by (6).

Case1: \( r = 0 \) in (6), then \( 0 \leq c \leq 0 \), thus \( c = 0 \).

Case2: \( r = 1 \) in (6), then \( 1 \leq c \leq L \), thus \( c \neq 0 \).

Case3: \( c = 0 \) in (6), then \( r \leq c \leq L \times r \), thus \( r = 0 \).

Case4: Let us prove “if \( c \neq 0 \), then \( r = 0 \).” For the propose of contradiction, assume \( r = 0 \), then by Case1 we have \( c = 0 \), which is a contradiction. Thus, \( r \) must be 1. \( \Box \)

According to Theorem 1, (5) can be replaced by

\[
q_{ij} \leq p(j) \leq F \times q_{ij}, \quad j = 1, 2, \ldots, S 
\]  

(7)

Based on decision variables \( Q \), the schedule length (SL) can be formulated as follows.

\[
SL = \sum_{j=1}^{S} q_{ij} 
\]  

(8)

We can avoid adding a dummy sink node into the MDFG by using (8). In addition, it is easier to model energy consumption problem in this way than the model proposed in [30].

**Objective function 1.** Performance optimization can be formulated as following objective function.

\[
\min \{SL\} 
\]  

(9)

**Dependency constraint.** For a given MDFG \( G, e(h,l) \in G \) indicates that operation \( i \) cannot be executed until operation \( h \) finished. The dependency constraint is formulated as

\[
\sum_{j=1}^{S} \sum_{k=1}^{F} j \times x_{i,j,k} \geq \sum_{j=1}^{S} \sum_{k=1}^{F} j \times x_{h,j,k} + L(h), \quad \forall e(h,l) \in G 
\]  

(10)

\( L(h) \) is the execution latency of operation \( h \). In our architectural model, we have \( \forall h \in G, L(h) = 1 \).

**Resource constraint.** In order to avoid resource conflicts, one time slot (one cycle) of a function unit can be used to execute only one operation.

\[
\sum_{i=1}^{N} x_{i,j,k} \leq 1, \quad j = 1, 2, \ldots, S, \quad k = 1, 2, \ldots, F 
\]  

(11)

**Variable assignment constraint.** To model the variable assignment, we define binary decision variables with two indices: \( M = \{m_{ij} \mid i = 1, 2, \ldots, N_v; j = N_m + 1, \ldots, F\} \). A decision variable \( m_{ij} = 1 \) only when variable \( i \) is assigned to memory module \( j \). Note that each variable can only be assigned to one memory module.

\[
\sum_{j=N_m+1}^{F} m_{ij} = 1, \quad i = 1, 2, \ldots, N_v 
\]  

(12)

Therefore, the memory module on which variable \( i \) is assigned can be stated in terms of \( m_{ij} \) as

\[
\text{mem}(i) = \sum_{j=N_m+1}^{F} j \times m_{ij}, \quad i = 1, 2, \ldots, N_v 
\]  

(13)

According to the architectural model, a memory access operation must be scheduled onto the MU connected to the memory module on which the corresponding variable has been assigned. The memory unit \( fu(v,i) \) allocated to an operation \( i \) which access the variable \( v \) is defined as

\[
f(u,v,i) = \sum_{j=N_m+1}^{F} j \times x_{i,j,k}, \quad \forall i \in [1, N] 
\]  

(14)

Then, the variable layout constraint is formulated as

\[
\text{mem}(v) = fu(v,i), \quad \forall v \in [1, N], \quad \forall i \in \left\{ l | \text{var}(i) = v \right\} 
\]  

(15)

**4.2. Memory energy consumption minimization**

We define binary variables with two indices: \( W = \{w_{jk} \mid j = 1, 2, \ldots, S; k = N_m + 1, N_m + 2, \ldots, F\} \) to represent the operating mode of a memory module at each control step. A decision variable \( w_{jk} = 1 \) only when memory module \( k \) is in standby operating mode in control step \( j \).

**Objective function 2.** According to the energy model, energy minimization can be formulated as following objective function.

\[
\min \left( N_{a \times c} + E_{a \times c} + N_m \times SL - E_s \times \sum_{j=1}^{S} \sum_{k=N_m+1}^{F} w_{jk} \right) 
\]  

(16)

We define some auxiliary variables and use a combination condition to determine the decision variables \( W \). We use an example as Fig. 5 to show how to determine energy mode of memory modules in each time slot. In this figure, column “ALU1” to column “M2” is a schedule table of an iteration body, column “cs2” represents time slot. “M1” and “M2” represent two memory access units. Considering memory module M2, column “wg(j,2)” column “wg(j,2)” help to explain why we need construct Eqs. (17)–(27) to determine “wg(j,2)”.

For the ease of explanation, we use the symbol grid(j, k), \( j \in [1, S], k \in [N_m + 1, F] \) to denote a grid in a schedule table. If there
is no operation assigned to grid(j,k) and step j is valid step (refer to (5)), then wig(j,k) is a valid idle grid. A function wig(j,k) is defined to present the state of grid(j,k).

\[
wig(j,k) = wwg(j,k) + (1 - q_j), \quad j \in [1,S], \; k \in [Na + 1,F]
\]

(17)

\[
wig(j,k) = \sum_{h=j}^{N} x_{i,j}(h,k)
\]

is a 0–1 function denotes the number of operation assigned to grid(j,k). According to (5) and (11), we know wig(j,k) is also a 0–1 function. wig(j,k) = 0 denotes grid(j,k) is a valid idle grid. We use Fig. 5 to explain Eq. (17). In Fig. 5, if wig(j,k) = 0, then grid(j,k) is possible to set to low power mode, but the final determination is related to the time cost of energy model and iteration.

It is well known that the energy and time consumption of a given application are dominated by loops. Therefore, we not only exploit intra-iteration energy saving, but also consider the potential energy saving across iterations. We construct two kinds of conditions to judge whether a grid(j,k) in standby mode or not.

We define a function \(wl^1(j,k)\) to present the state of three successive grids from grid(1,k).

\[
wl^1(j,k) = \begin{cases} 
\frac{1}{2} \sum_{h=j}^{S} wig(h,k), & \text{if } j \in [1,S-2], k \in [Na+1,F] \\
1, & \text{if } j \in [S-1,S], k \in [Na+1,F] 
\end{cases}
\]

(18)

\(wl^1(j,k)\) is an integer value between 0 to 3. \(wl^1(j,k) = 0\) represents that grid(j,k) can be in standby mode, i.e. if the successive valid idle period (from step j) is longer than the time of reactivation, then we set grid(j,k) to standby mode. For example in Fig. 5, according to the column “\(wl^1(j,2)\),” we can determine that grid(1,M2) and grid(2,M2) are in standby mode. The following condition is applied to determine the intra-iteration energy savings.

**Condition 1.** For any grid(j,k), it is in standby mode if and only if \(wl^1(j,k) = 0\).

We define binary decision variables: \(W^1 = \{w^1_{h,j}, j = 1, 2, \ldots, S; k = Na + 1, Na + 2, \ldots, F\}\) to present the operating mode of each grid(j,k) determined by Condition 1.

\[
w^1(j,k) = \begin{cases} 
1 & \text{if } wl^1(j,k) = 0 \\
0 & \text{if } wl^1(j,k) = 1, 2, 3
\end{cases}
\]

(19)

If \(w^1_{h,j} = 1\) then grid(j,k) is in standby mode. We show that this type of nonlinear condition can be modeled with linear constraints via the following theorem.

**Theorem 2.** Let \(c\) be an integer variable whose value is between 0 and \(L\), and \(r\) be a binary variable. The nonlinear condition: “\(c = 0\) if and only if \(r = 1\)” can be modeled as the following linear form:

\[
1 - r \leq c \leq (1 - r)
\]

(20)

**Proof.** Let us prove the following statements are correctly modeled by (20).

Case1: If \(r = 1\) in (20), then \(0 \leq c \leq 0\), thus \(c = 0\).

Case2: If \(r = 0\) in (20), then \(1 \leq c \leq 1\), thus \(c = 0\).

Case3: If \(c = 0\) in (20), then \(1 - r \leq 0\), thus \(r = 1\).

Case4: Let us prove “if \(c \neq 0\), then \(r = 0\).” For the propose of contradiction, let us assume \(r = 1\), then by Case1 we have \(c = 0\), which conflicts with the condition \(c \neq 0\). Therefore, if \(c \neq 0\), then \(r = 0\) must be.

According to Theorem 2, (19) can be replaced by

\[
1 - w^1_{h,j} \leq w^1(j,k) \leq 3 \times (1 - w^1_{h,j})
\]

(21)

In a loop, the operating modes of a memory module in the last two control steps of an iteration are related to the memory accesses of the first two control steps of next iteration. For example in Fig. 5, the operating modes of grid(1,M2) and grid(2,M2) can be switched into standby mode, because grid(1,M2) and grid(2,M2) are valid idle grids. However, the operating modes of grids in last two steps cannot be determined by Condition 1. To consider the inter-iteration energy saving, we define a function \(wl^2(j,k)\) to represent the operating mode of the last control step.

\[
w^2(j,k) = \begin{cases} 
wig(j,k) + \sum_{h=1}^{S} wig(h,k) + q_{j+1}, & \text{if } j \in [1,S-1], \\
1, & \text{if } j = S
\end{cases}
\]

(22)

where \(k \in [Na+1,F]\), \(q_{j+1}\) is used to judge the real schedule length. \(wl^2(j,k)\) is an integer value between 0 to 4.

**Condition 2.** For any grid(j,k), it is in standby mode if and only if \(wl^2(j,k) = 0\).

We define binary decision variables: \(W^2 = \{w^2_{h,j}, j = 1, 2, \ldots, S; k = Na + 1, Na + 2, \ldots, F\}\) to present the operating mode of each determined by Condition 2. A decision variable \(w^2_{h,j}\) can be stated in terms of \(wl^2(j,k)\).

\[
w^2_{h,j} = \begin{cases} 
1 & \text{if } wl^2(j,k) = 0 \\
0 & \text{if } wl^2(j,k) = 1, 2, 3, 4
\end{cases}
\]

(23)

\(w^2_{h,j} = 1\) denotes that grid(j,k) is in standby mode. According to Theorem 2, (23) is equivalent to

\[
1 - w^2_{h,j} \leq wl^2(j,k) \leq 4 \times (1 - w^2_{h,j})
\]

(24)

Similarly, we define a function \(wl^3(j,k)\) to formulate the operating mode of grid(i,j) in the second to last step.

\[
w^3(j,k) = \begin{cases} 
wig(j,k) + \sum_{h=j}^{S} wig(h,k) + w_{j+2}, & \text{if } j \in [1,S-2], \\
1, & \text{if } j \in [S-1,5]
\end{cases}
\]

(25)

where \(k \in [Na+1,F]\). \(wl^3(j,k)\) is an integer value between 0 and 4.
Condition 3. For any grid \((j,k)\), it is in standby mode if and only if \(w^I(j,k) = 0\).

We define binary decision variables: \(W^3 = \{w^3_{j,k} : j = 1, 2, \ldots, S; k = N_0 + 1, N_0 + 2, \ldots, F\}\) to present memory models in which operating mode. According to Theorem 2 and Condition 3, decision variable \(w^3_{j,k}\) should satisfy the following constraint.

\[
1 - w^3_{j,k} \leq w^I(j,k) \leq 4 \times (1 - w^3_{j,k})
\]  

(26)

Finally, a decision variable \(w_{j,k}\) can be stated as

\[
w_{j,k} = w^1_{j,k} + w^2_{j,k} + w^3_{j,k}
\]

(27)

The example in Fig. 5 demonstrated how to determine energy mode of one memory with specific schedule table. Column "\(wig(j,2)\)" add "1 – \(q_j\)" and get the column "\(wig(j,2)\)" by using Eq. (17). According to column "\(wig(j,2)\)" and Eq. (18), we can get values of "\(w^I(j,k)\)". Then according to Eq. (21) we can get values of decision variable \(w^3(j,k)\). With the similar way, we can get value of column "\(w^2(j,k)\)" and then "\(w^1(j,k)\)". Finally, according to Eq. (27), we can determine the decision variable "\(w(j,k)\)".

For the real-time application, we can add schedule length constraint to meet the performance requirement.

\[SL \leq \text{deadline}\]

(28)

In this section, we give the ILP model for energy optimization and explain how to generate this model by an example. This ILP model is related to energy model of multi-modules memory. However, it can be used in similar scenarios with modified parameters.

4.3. Register constraint

To improve instruction level parallelism, the values of variables need to be loaded into registers as soon as possible. On the other hand, to obtain longer consecutive idle time, the results of operations need to be stored back to memory as late as possible. To achieve these two goals, we need more registers for saving temporary values. However, the number of registers available for memory access operations is very limited in DSP processor. When there are more live variables than available registers, register spilling will occur. The performance and energy penalties of register spilling are very high. Therefore, the limited number of registers must be taken into account in the instruction scheduling process.

Register allocation for load operation. For a given MDFG \(G\), the edge set \(E^1 = \{e(u,v) : e(u,v) \in E, t(u) = M, t(v) = A\}\) denotes that the starting point of an edge is a load operation and the ending point of the edge is an ALU operation. We define a function \(R^1(l,u)\) to represent whether at step \(l\) a register is in use by the operand loaded by operation \(u\).

\[
R^1(l,u) = \sum_{j=1}^{l} \sum_{k=1}^{C_0} (x_{uj,k} - x_{uj,k}), \quad l = 1, 2, \ldots, S, \quad \forall e(u,v) \in E^1
\]

(29)

Then, we use the function \(Reg^1(l)\) to represent the number of registers allocated for all the load operations in step \(l\).

\[
Reg^1(l) = \sum_{u=1}^{N} R^1(l,u), \quad l \in [1,S], \quad \forall e(u,v) \in E^1
\]

(30)

Register allocation for store operation. Similarly, another edge set \(E^2 = \{e(u,v) : e(u,v) \in E, t(u) = A, t(v) = M\}\) denotes that the starting point of \(e(u,v)\) is an ALU operation and the ending point is a store operation. We define \(R^2(l,u)\) to represent whether at step \(l\) a register is in use by the operand produced by \(u\), waiting to be stored back to memory.

\[
R^1(l,u) = \sum_{j=1}^{l} \sum_{k=1}^{C_0} (x_{uj,k} - x_{uj,k}), \quad l = 1, 2, \ldots, S, \quad \forall e(u,v) \in E^1
\]

The number of registers allocated for saving values to be stored backed at step \(l\) can be stated as follows.

\[
Reg^2(l) = \sum_{u=1}^{N} R^2(l,u), \quad l \in [1,S], \quad \forall e(u,v) \in E^2
\]

(32)

Register number constraint. Finally, the register number constraint can be formulated in terms of \(Reg^1(l)\) and \(Reg^2(l)\).

\[
Reg^1(l) + Reg^2(l) \leq N, \quad l = 1, 2, \ldots, S
\]

(33)

5. Reduce computation time of the basic ILP model

In this section, we present two methods to improve the execution efficiency of the basic ILP model proposed in Section 4. The first one is called the optimality guarantee method, from which we can get the optimal solution in a reasonable amount of time. The second is an approximate method, by which the computation time can be reduced drastically without sacrificing much optimality.

5.1. Optimal guarantee method

A mobility window of operation \(v\), which is denoted by \(MW(v)\), is the set of control steps in which operation \(v\) can be executed. Tightening the mobility window of each operation is similar to reducing the number of decision variables and cutting off some fractional solution from the feasible region of LP-relaxation. Consequently, the solution time of the proposed ILP model can be significantly reduced. For instance, if the mobility window of operation \(v\) is \(MW(v) = \{2, 3\}\), only the decision variables \(x_{v,2,2}\) and \(x_{v,3,2}\) are binary variables, other variables \(x_{v,2,1}\) and \(x_{v,3,1}\) are binary variables, other variables \(x_{v,2,3}\) and \(x_{v,3,3}\) can be set to 0 directly. If the upper bound of schedule length is 100 (\(S = 100\)) and the number of function units is \(4(k = 4)\), the number of decision variables is 400 without mobility window tightening, but it is reduced to only 8 with tightening.

Conventionally, the mobility window is always derived from as soon as possible (ASAP) and as late as possible (ALAP) scheduling algorithms, in which the number of resources is unlimited. We can generate a tighter mobility window by considering hardware resource constraints and the inherent dependencies in the input MDFG.

Proposition 1. Given an MDFG \(G\), for an operation \(v\) of type \(k\), let \(\text{ancestors}(v,k)\) denote the number of operations of type \(k\) that are ancestors of \(v\), and \(\text{children}(v,k)\) denote the number of operations of type \(k\) that are children of \(v\). \(\text{res}(k)\) denotes the number of functional units of type \(k\). ASAP\((v)\) and ALAP\((v)\) are the starting step of \(v\) in ASAP and ALAP schedule, \(DL(v)\) represents the deadline of the start time of operation \(v\). The lower bound and upper bound of \(MW(v)\) are:

\[
L_{MW}(v) = \max \left\{ \text{ASAP}(v), \frac{\text{ancestors}(v,k)}{\text{res}(k)} + 1 \right\}
\]

(34)

\[
U_{MW}(v) = \min \left\{ \text{ALAP}(v), DL(v) - \frac{\text{children}(v,k)}{\text{res}(k)} - 1 \right\}
\]

(35)

5.2. Approximation method

Variable neighborhood search (VNS) is a recent meta-heuristic for building heuristic to solve a variety of combinatorial and global optimization problems [31]. The basic idea is to systematically change the neighborhood of the variable within a local search framework to achieve the best solution. We employ this idea to
design a heuristic algorithm based on the basic ILP model. It is called the VNS-ILP algorithm shown in Algorithm 5.1. When the search radius \( r \) is small, the computation time of basic ILP model is very short. When the search radius is increased, the optimality of the schedule can be improved at the cost of more computation time. If the search radius is sufficiently large, the global optimal solution can be achieved. In order to improve the practicality, we introduce a parameter stalling time (ST) to limit the computation time.

Algorithm 5.1. VNS based approximate ILP algorithm

\begin{verbatim}
Input: (1) MDFG: G, (2) search radius \( R \), (3) Stalling Time: ST
Output: The best schedule respect to the objective function
1: \( S_{best} \leftarrow \) random variable partion and list scheduling (G);
2: Initialize the search radius \( r \rightarrow 1 \);
3: while \((\text{running time} < \text{ST}) \) \( \text{and} \) \((\text{exist} \ MW(i) \leq MW_{\text{max}}(i))\) \( \text{and} \ (r < R) \) \( \text{do} \)
   4: Generate_MW(G, r, S_{best})
   5: Add mobility window constraints to the original ILP model
   6: \( S_{new} \leftarrow \) solving the new ILP model
   7: if objective_function(\( S_{new} \)) < objective_function(\( S_{best} \))
      then
         8: \( S_{best} \leftarrow S_{new} \)
         9: \( r \rightarrow r - 1 \)
      else
         10: \( r \rightarrow r + 1 \)
      end if
   12: end if
13: end while
14: return \( S_{best} \)
\end{verbatim}

In the 4th line, we generate the mobility window for each operation based on the reference schedule and search radius \( r \). For all the decision variables related to an operation \( v \), if the index \( j \) is out of \( MW(v) \) then we let them equal to 0. Note that the optimal solution derived from VNS-ILP algorithm may not be a global solution of the basic ILP model. In the 7th line, the "Objective Function" can be one of the functions in Section 4 ((9) and (16)). The maximal mobility window for operation \( v \) is denoted by \( MW_{\text{max}}(v) \), which can be computed by (34) and (35). For all operations \( v \), if \( MW(v) \geq MW_{\text{max}}(v) \) then the global optimal solution can be achieved. There is no need to extend the search radius further and the while loop can be terminated. Nevertheless, it is not worthwhile to search a long time to get the global optimal solution, especially for the large scale problem. In practice, the computation time can be controlled by the ST parameter. Thus, VNS-ILP algorithm is more practical than the pure ILP method.

6. Experiment

We conduct some experiments on a set of representative DSP kernels [32] to evaluate the effectiveness of the proposed optimization techniques. To prepare the input data of our simulation experiments, we add a pass to the compiler experimental framework Trimaran [33] to transform C programs into MDFGs. Note that in these benchmarks the energy and time consumptions are dominated by the loops, so we only consider the MDFG of the loop bodies. Table 2 lists the number of operations and variables in these MDFGs. We use a commercial ILP solver CPLEX [34] to solve our ILP model. The input graph processing and the VNS-ILP algorithm are implemented in the C++ language. All the experiments are run on a 2.8 GHz Pentium 4 PC with 2GB of memory.

<table>
<thead>
<tr>
<th>Bench</th>
<th>Description</th>
<th>#Ops</th>
<th>#Vars</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>Least_mean_square</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>B2</td>
<td>N_complex_updates</td>
<td>24</td>
<td>5</td>
</tr>
<tr>
<td>B3</td>
<td>Fir2dim</td>
<td>24</td>
<td>6</td>
</tr>
<tr>
<td>B4</td>
<td>All-pole lattice filter</td>
<td>30</td>
<td>6</td>
</tr>
<tr>
<td>B5</td>
<td>Differential Equation</td>
<td>33</td>
<td>7</td>
</tr>
<tr>
<td>B6</td>
<td>Biquad_N_sections</td>
<td>34</td>
<td>5</td>
</tr>
<tr>
<td>B7</td>
<td>4-stage lattice filter</td>
<td>52</td>
<td>9</td>
</tr>
<tr>
<td>B8</td>
<td>Elliptic filter</td>
<td>69</td>
<td>8</td>
</tr>
</tbody>
</table>

6.1. Comparison of schedule lengths

In this section, we compare the schedule lengths generated by our PEOS method with the IG [6] and VIG [7] methods. We assume that the target processor consists of two ALUs, two data memory modules and four registers. The energy consumption is not taken into account. The experimental results and comparisons are shown in Table 3, where the "SL" columns represent the schedule length in control step. The percentage of schedule length reductions by PEOS compared with the IG and VIG method are shown in columns "Imp1" and "Imp2," respectively. All the results generated by PEOS are optimal. The average improvements reflect that high performance schedule can only be achieved by phase coupled optimization techniques.

The computation time of our PEOS method is another concern that needs to be addressed. We take the schedule lengths derived from the IG method as the upper bounds \( S \) (see Section 4.1). Column "Time1" lists the running time of the basic ILP model for each benchmark. Column "Time2" lists the running time of the basic ILP model with tightened mobility window (see Section 5.1). From these results we can see that computation time of our PEOS method for large-scale benchmarks are relatively long, e.g. benchmark B8 taxes about 20 min. Adding the tighter mobility window, the running time can be reduced drastically. Comparing the running times of benchmarks B5–B8, the execution efficiency is improved about 18 times on average. This experiment confirms that our ILP-based method is very promising for achieving the optimal schedule within a short computation time.

6.2. Comparison of energy consumptions

We conduct two experiments to illustrate the energy effectiveness of our ILP-based method. The target processor configuration is the same as that in the last experiment. The energy model is the same as that in Section 2.1.

Optimal energy consumption. In this experiment, the timing constraint is not taken into account. We compare the energy consumptions of our PEOS method with the IG [6] and VPIS [16] methods.

Table 2

<table>
<thead>
<tr>
<th>Characteristics of the benchmarks.</th>
</tr>
</thead>
<tbody>
<tr>
<td>No.</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>B1</td>
</tr>
<tr>
<td>B2</td>
</tr>
<tr>
<td>B3</td>
</tr>
<tr>
<td>B4</td>
</tr>
<tr>
<td>B5</td>
</tr>
<tr>
<td>B6</td>
</tr>
<tr>
<td>B7</td>
</tr>
<tr>
<td>B8</td>
</tr>
</tbody>
</table>

Table 3

<table>
<thead>
<tr>
<th>Pure schedule length comparisons.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bench</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>SL (cs)</td>
</tr>
<tr>
<td>B1</td>
</tr>
<tr>
<td>B2</td>
</tr>
<tr>
<td>B3</td>
</tr>
<tr>
<td>B4</td>
</tr>
<tr>
<td>B5</td>
</tr>
<tr>
<td>B6</td>
</tr>
<tr>
<td>B7</td>
</tr>
<tr>
<td>B8</td>
</tr>
</tbody>
</table>

Average: 25.8 15.5
The results are shown in Table 4, which reveal that our PEOS method can save 31.0% and 21.8% more energy than the IG and VPIS methods on average, respectively. In addition, all the results derived from our PEOS method are optimal solutions. The “Time” column represents time consumption of PEOS with tighter mobility windows.

The energy saving are mainly due to the following two reasons. (1) Our PEOS method can lump the dispersed idle cycles into consecutive idle periods as long as possible without extending the schedule length. On the other hand, the VPIS method achieves longer consecutive idle period by adding extra control steps, increasing the static energy consumption. (2) In our approach, variable assignments are also energy aware to keep memory modules in standby mode for a longer period of time. On the other hand, the IG and VPIS methods always partition variables into different memory modules in the first step without sufficient consideration to energy. To achieve the schedule with optimal energy, for a given MDFG G, we take |V| as the upper bound S. Consequently, for the large scale benchmarks like B8, the computation time is much longer. However, it can find the lower bound of energy consumption, which is useful for making energy budgets and evaluating the heuristic energy saving algorithms.

**Optimal energy consumption with timing constraint.** In practice, timing constraints are critical prerequisites for energy-aware scheduling. In the second experiment, we evaluate the effectiveness of our PEOS method on energy saving without missing the schedule deadline. Table 5 compares our PEOS method with the VPIS method in terms of energy consumption and schedule length. We take the schedule length derived from the IG algorithm as the deadline of each benchmark (see Table 3).

The percentage of reduction on energy consumption and schedule length compared to VPIS are shown in columns “Imp1” and “Imp2,” respectively. From the experimental results, we can see that our PEOS method can, on average, achieve 23.1% more energy savings while reducing the schedule length by 20.9%. It confirms that our proposed method can achieve a good tradeoff between energy and performance. In some cases such as B3–B7, the schedules generated by the PEOS method are optimal with respect to both energy and schedule length. Since the deadline is given, the computation time is much shorter than the pure energy minimization case. Note that the tighten mobility window constraints had been added to the basic ILP model of PEOS.

### 6.3. Evaluation of the approximation method

We conduct experiments on benchmarks B7 and B8 to evaluate the effectiveness of the VNS-ILP algorithm. The curves in Figs. 6 and 7 shows the energy consumption and computation time with increasing search radius. Energy consumptions are normalized to the maximum energy, time costs are normalized to the time when the optimal energy are obtained by PEOS. All energy and time values are between 0 and 1.

We set the stalling time to be 2 min. For the benchmark B7, when search radius \( r = 4 \) the optimal energy schedule can be achieved, but the computation time exceeds the stalling time. If \( r = 3 \), the energy consumption is only 6.2% higher than the optimal energy value, but the computation time is only 46 seconds. For the benchmark B8, the optimal energy schedule is achieved when \( r = 5 \), but the computation time is about 20 min. When \( r = 3 \), the ILP model can be solved within 2 min. The energy consumption is only

### Table 4

<table>
<thead>
<tr>
<th>Bench</th>
<th>IG (Energy)</th>
<th>VPIS (Energy)</th>
<th>PEOS (Energy)</th>
<th>IG (Time)</th>
<th>VPIS (Time)</th>
<th>PEOS (Time)</th>
<th>Imp1 (%)</th>
<th>Imp2 (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>13.53</td>
<td>12.94</td>
<td>9.51</td>
<td>4</td>
<td>29.9</td>
<td>26.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B2</td>
<td>23.29</td>
<td>20.63</td>
<td>16.80</td>
<td>7</td>
<td>27.9</td>
<td>18.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B3</td>
<td>24.37</td>
<td>23.37</td>
<td>17.39</td>
<td>15</td>
<td>28.6</td>
<td>25.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B4</td>
<td>28.66</td>
<td>23.39</td>
<td>19.02</td>
<td>116</td>
<td>33.6</td>
<td>18.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B5</td>
<td>35.02</td>
<td>28.24</td>
<td>21.43</td>
<td>255</td>
<td>38.8</td>
<td>24.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B6</td>
<td>36.52</td>
<td>31.18</td>
<td>27.59</td>
<td>37</td>
<td>22.5</td>
<td>11.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B7</td>
<td>35.62</td>
<td>45.49</td>
<td>32.00</td>
<td>553</td>
<td>36.5</td>
<td>29.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B8</td>
<td>58.26</td>
<td>50.50</td>
<td>40.52</td>
<td>1380</td>
<td>30.4</td>
<td>19.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>31.0</td>
<td>21.8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 5

<table>
<thead>
<tr>
<th>Bench</th>
<th>VPIS (Energy)</th>
<th>PEOS (Energy)</th>
<th>VPIS (Time)</th>
<th>PEOS (Time)</th>
<th>Imp1 (%)</th>
<th>Imp2 (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>12.94</td>
<td>10.28</td>
<td>7</td>
<td>2</td>
<td>20.6</td>
<td>12.5</td>
</tr>
<tr>
<td>B2</td>
<td>20.63</td>
<td>17.14</td>
<td>11</td>
<td>5</td>
<td>16.9</td>
<td>15.4</td>
</tr>
<tr>
<td>B3</td>
<td>23.76</td>
<td>17.39</td>
<td>12</td>
<td>11</td>
<td>26.8</td>
<td>25.0</td>
</tr>
<tr>
<td>B4</td>
<td>26.22</td>
<td>19.02</td>
<td>14</td>
<td>43</td>
<td>27.5</td>
<td>26.3</td>
</tr>
<tr>
<td>B5</td>
<td>28.24</td>
<td>21.43</td>
<td>14</td>
<td>98</td>
<td>24.1</td>
<td>30.0</td>
</tr>
<tr>
<td>B6</td>
<td>33.18</td>
<td>27.59</td>
<td>16</td>
<td>20</td>
<td>16.8</td>
<td>27.3</td>
</tr>
<tr>
<td>B7</td>
<td>47.49</td>
<td>32.00</td>
<td>23</td>
<td>218</td>
<td>32.4</td>
<td>28.1</td>
</tr>
<tr>
<td>B8</td>
<td>50.50</td>
<td>40.52</td>
<td>34</td>
<td>524</td>
<td>19.8</td>
<td>2.86</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>23.1</td>
<td>20.9</td>
</tr>
</tbody>
</table>
7. Conclusions

In this paper, we proposed an ILP-based technique to achieve the optimal schedule with respect to energy consumption and performance. In the PEOS ILP framework, the operating mode transition for energy saving and register constraints are exactly formulated. It can address the instruction scheduling and variable assignment problems simultaneously, which is superior to conventional phased decoupled methods. A special effort is made to explore the inter-iteration energy saving. In addition, we proposed two efficiency optimization techniques to improve the practicability of the basic PEOS model. Experimental results show that all the proposed techniques are very promising. We are currently working on finding new application specific cutting planes to accelerate the ILP model solving further.

Acknowledgements

This work is partially supported by the NSFC 61071061 and the Univ. of Kentucky Start Up Fund; NSF CNS-1015802, Texas NHARP 009741-0020-2009, HK GRF 123609, NSFC 60728206, Changjiang Honorary Chair Professor Scholarship.

References

Edwin Hsing-Mean Sha received Ph.D. degree from the Department of Computer Science, Princeton University, USA in 1992. From August 1992 to August 2000, he was with the Department of Computer Science and Engineering at University of Notre Dame, USA. Since 2000, he has been a tenured full professor in the Department of Computer Science at the University of Texas at Dallas. He has published more than 270 research papers in refereed conferences and journals. He has served as an editor for many journals, and as program committee and Chairs for numerous international conferences. He received Teaching Award, Microsoft Trustworthy Computing Curriculum Award, NSF CAREER Award, and NSFC Overseas Distinguished Young Scholar Award. He was awarded as Chiang Jiang Honorary Chair Professorship and China Thousand Talents Program associated with Hunan University, China. His web page can be found in http://www.utdallas.edu/~edsha.

Dr. Qingfeng Zhuge received her Ph.D. from the Department of Computer Science at the University of Texas at Dallas in 2003. She obtained her BS and MS degrees in Electronics Engineering from Fudan University, Shanghai, China. She is currently a professor at Hunan University, China. She received Best Ph.D. Dissertation Award in 2003. She has published more than 60 research articles in premier journals and conferences. Her research interests include parallel architectures, embedded systems, supply-chain management, real-time systems, optimization algorithms, compilers, and scheduling.