Memory Access Driven Storage Assignment for Variables in Embedded System Design

Yoonseo Choi and Taewhan Kim
Department of Electrical Engineering & Computer Science and Advanced Information Technology Research Center (AITrc)
Korea Advanced Institute of Science and Technology, KOREA
yschoi@jupiter.kaist.ac.kr tkim@cs.kaist.ac.kr

Abstract – It has been reported and verified in many design experiences that a judicious utilization of the page/burst access modes supported by DRAMs contributes a great reduction in not only the DRAM access latency but also DRAM’s energy consumption. Recently, researchers showed that a careful arrangement of data variables in memory directly leads to a maximum utilization of the page/burst access modes for the variable accesses, but unfortunately, found that the problems are not tractable, consequently, resorting to simple (e.g., greedy) heuristic solutions to the problems. In this paper, to improve the quality of existing solutions, we propose a new storage assignment technique, called zone alignment, for variables, which effectively exploits an efficient 0-1 ILP formulation and the temporal locality of variables’ accesses in code.

I. INTRODUCTION
In embedded systems, memory is one of the major sources of performance bottleneck and power consumption [3]. It has been known that most of modern DRAMs used in embedded system design support efficient memory access modes. In particular, the page and burst access modes are extensively supported in DRAMs [4], [5]. In general, the latency of random access is much longer than that of page/burst access. Further, memory operates in lower power per bit in page/burst mode than in random access mode. For example, IBM’s Cu-11 Embedded DRAM [4] macro supports random access mode of 10ns and page mode access of 5ns at worst. It has active current of 60mA/MB roughly in random cycle and 13mA/MB in page cycle. Consequently, a full exploitation of these memory access modes is very necessary to alleviate the performance bottleneck caused by the delay of memory accesses. Panda et al. [1] modeled a number of realistic page access modes in DRAMs and proposed an algorithm for arranging non-array variables to memory and organizing array variables. The formulation of the problem of arranging non-array variables in a memory is analogous to that in [2]. Khare et al. [6] extended the work in [1] to support the burst mode in a dual-memory architecture focusing mainly on an efficient interleaving of memory accesses. Grun et al. [7] proposed compiling technique to exploit detailed timing information of the memories. They also present an approach [8] of extracting, analyzing and clustering the most active memory access patterns to customize memory architecture. Ayukawa et al. [9] proposed an access sequence control scheme using an hardware for relieving the page-miss penalty in random access mode. Hettiaratchi et al. [10] proposed an address assignment technique for array variables to reduce energy consumption on memory accesses by maximizing, so called, page mode accesses. They formulated the problem into a multi-way graph partitioning and solved it using a Kernighan-Lin based partitioning algorithm. Finally, the authors of [11] confirmed that the problems of maximizing page/burst mode accesses by variable assignment to DRAM memory are NP-hard, and proposed two greedy heuristic solutions to the problems. In this paper, to complement the prior work [1], [2], [6], [7], [8], [9], [10], [11], we try to tackle the problems optimally. Specifically, we develop an efficient ILP based technique, called zone alignment, for storage assignment of variables to maximally utilize page/burst mode accesses.

Acknowledgment: This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc).

II. PRELIMINARIES AND MOTIVATIONS
For a sequence of variable references \(a_1, a_2, \ldots, a_t, \ldots\), and an assignment of the variables to the memory, let us denote \(v(a_i)\) the variable referenced by \(a_i\). Further, let \(f_{page}(v(a_i))\) and \(f_{addr}(v(a_i))\) be the page and relative location of the memory at which variable \(v(a_i)\) is located, respectively. Formally, we can define page and burst accesses as: (1) when normal mode and page mode are available, the access \(a_i\) is called a page access if and only if \(f_{page}(v(a_i)) = f_{page}(v(a_{i-1}))\); (2) when normal mode and burst mode are available, the access \(a_i\) is called a burst access\(^1\) if and only if \(f_{addr}(v(a_i)) = f_{page}(v(a_{i-1}))\) and \(f_{addr}(v(a_{i-1})) + 1\). Note that a page access not only has much shorter access delay, but also consumes much less accessing energy than an initial normal access.

![Diagram](image-url)

Fig. 1. An example illustrating the effects of memory layout on the number of page accesses.

One of the most effective ways to reduce the memory access latency and energy consumption using page mode is to find the (relative) placement of variables in memory (i.e., memory layout for variables) which offers the maximum number of page accesses. Figure 1 illustrates how different memory layouts affect the memory access latency and energy consumption. Note that the memory layout in Figure 1(b) results in 16.7% reduction in memory access latency, and 29.3% in energy consumption compared with that of Figure 1(a).

III. MEMORY ALIGNMENT FOR MAXIMIZING PAGE ACCESSES

A. Problem Definition
The optimization problem is, given a sequence of variable accesses, to find a memory layout with maximum number of page accesses (e.g., ibm Cu-11 Embedded DRAM [4]). We call it the MLP problem (Memory Layout with Page mode). An instance of MLP is characterized by \((S, V, m)\) where \(S\) is a variable sequence to be accessed, \(V\) is the set of variables in \(S\), and \(m\) is the page size, partitioning the variables in \(V\) into disjoint groups, each of which has \(m\) or less than \(m\) variables.

Definition 3.1: The access graph of \(S\) is a multigraph \(G(V,E)\) where node set \(V\) is the set of variables in \(S\) and there are \(n\) edges between

\(^1\)There are a number of techniques for implementing burst modes. Our definition is an abstraction of them.
two nodes \(v_i\) and \(v_j\) if and only if \(v_i\) and \(v_j\) are adjacent to each other in \(S\) exactly \(n\) times.

Figure 2(a) shows the access graph as in Definition 3.1 for a sequence of variable accesses. For example, the access graph has three edges between nodes \(a\) and \(c\) since variables \(a\) and \(c\) are adjacent to each other three times in \(S\).

\[
\begin{array}{c}
\text{(a)} \\
\text{S: ace b c ad e f} \\
\text{m (i.e., page size) = 3}
\end{array}
\]

\[
\begin{array}{c}
\text{(b)} \\
\text{(c)}
\end{array}
\]

Fig. 2. A sequence of variable accesses and (a) its access graph representation; (b) the optimal graph partitioning; (c) its implied memory layout.

We define a problem of graph partitioning, \(\Pi^m_{G(V,E)}\), of multigraph \(G(V,E)\), as partitioning \(V\) into \(t\) disjoint subsets \(V_1, V_2, \ldots, V_t\), each containing at most \(m\) vertices with the objective of maximizing the quantity of

\[
g(\Pi^m_{G(V,E)}) = \sum_{(v_i,v_j) \in E, \ v_i,v_j \in V} w(v_i,v_j) \tag{1}
\]

where \(w(v_i,v_j)\) represents the number of edges between \(v_i\) and \(v_j\) in \(G\).

**Theorem 3.1:** [11] The MLP problem of \((S,V,m)\) is equivalent to the graph partitioning problem of \(G(V,E)\).

According to Theorem 3.1, every memory layout implied by an optimal graph partitioning of the access graph is optimal. For example, when \(m = 3\), Figure 2(b) shows an optimal graph partitioning whose gain is 8 (i.e., the number of heavy edges) and Figure 2(c) shows its implied memory layout, leading to 8 page accesses, which is the same as the gain of the partitioning. Thus, we approach to solve the MLP by deriving the corresponding access graph and finding graph partitioning with maximum value of \(g(\cdot)\).

**Theorem 3.2:** [11] The MLP problem is NP-hard.

**B. The Proposed Technique**

We devise a divide and conquer technique, called \(\text{zone alignment}\), for solving large size problem instances of the MLP. \(\text{Zone alignment}\) is an iterative technique. At each iteration, it extracts a subsequence of variable accesses (called \(\text{access zone}\)) from the input access sequence by analyzing the lifetimes of variables in the sequence, and applies our 0-1 ILP formulation to the access graph corresponding to the \(\text{access zone}\) to produce an optimal alignment of variables to memory. The process then repeats until all the variables in the access sequence are placed to the pages of memory. The effectiveness of \(\text{zone alignment}\) is supported by the following two facts: (1) It is generally true that a variable’s lifetime does not span the whole access sequence in code, and rather it is local. Consequently, the dividing strategy used in \(\text{zone alignment}\) can fully exploit the temporal locality of variables’ accesses; (2) \(\text{Zone alignment}\) solves each \(\text{access zone}\) optimally.

Suppose that we have divided an input access sequence into a number of \(\text{access zones}\) of manageable size. (The issue of determining \(\text{access zones}\) will be discussed later in this section.) Let \(Z\) be one of the \(\text{access zones}\), spanning from \(c_{\text{start}}\) to \(c_{\text{end}}\) of the access sequence. A few definitions and our modified ILP formulation for solving the storage alignment of the variables in \(Z\) are described in the following:

- \(V_{\text{in}}\): the set of variables whose lifetimes expire between the time of access \(c_{\text{start}}\) and \(c_{\text{end}}\). Precisely speaking, ends in the interval of \([c_{\text{start}}, c_{\text{end}}]\), and whose storage locations (i.e., pages) have not been determined yet.
- \(V_{\text{cross}}\): the set of variables whose lifetimes start before (or at) \(c_{\text{end}}\) and ends after \(c_{\text{end}}\), and whose storage locations have not yet been determined.
- \(V_{\text{zone}}\): \(V_{\text{in}} \cup V_{\text{cross}}\), which are the set of variables that are considered for storage assignment in the current \(\text{access zone}\) \(Z\).

Note that our ILP based \(\text{zone alignment}\) for current \(\text{access zone}\) \(Z\) should assign storage locations for all variables in \(V_{\text{in}}\), but assigns storage locations selectively for the variables in \(V_{\text{cross}}\) because only a partial information of consecutive accesses in access sequence with respect to the variables in \(V_{\text{cross}}\) has been known at the current \(Z\). Our ILP formulation attempts to select the variables in \(V_{\text{cross}}\) and assign pages to them based on the partial access information associated with the variables, and the assignment of pages to the remaining unselected variables will be done at the application of \(\text{zone alignment}\) to the succeeding \(\text{access zones}\) when the access information of the variables are sufficiently available.

Given an \(\text{access zone}\) \(Z\) and the access graph \(G(V,E)\), the problem we want to solve is to find partition \(\Pi^m_{G(V,E)}\) \(\{V_1,V_2,\ldots,V_{\lceil|V_{\text{zone}}|/m\rceil}\}\) of \(V_{\text{zone}}\) that maximizes the gain \(g(\cdot)\) as in Eq.(1). The 0-1 variables used in our ILP formulation are:

\[
x_k^i = \begin{cases} 1, & \text{if } v_i \in V_k \\ 0, & \text{otherwise.} \end{cases} \tag{2}
\]

\[
y_k = \begin{cases} 1, & \text{if } v_i \in V_k; \\ 0, & \text{otherwise.} \end{cases} \tag{3}
\]

The ILP formulation for access \(\text{zone}\) \(Z\) as is follows:

Maximize \[
\sum_{k=1}^{\lceil|V_{\text{zone}}|/m\rceil} \sum_{v_i \in V_{\text{zone}}} w_{ij} \cdot x_k^i 
\]

subject to,

\[
x_k^i - x_{k+1}^i = 0, \quad v_i, v_j \in V_{\text{zone}}, \quad k = 1, \ldots, \lceil|V_{\text{zone}}|/m\rceil \tag{4}
\]

\[
\sum_{v_i \in V_{\text{in}}} y_k^i = 1, \quad v_i \in V_{\text{in}} \tag{5}
\]

\[
\sum_{v_i \in V_{\text{cross}}} y_k^i \leq 1, \quad v_i \in V_{\text{cross}} \tag{6}
\]

\[
\sum_{v_i \in V_{\text{zone}}} \sum_{k=1}^{\lceil|V_{\text{zone}}|/m\rceil} y_k^i \leq \text{cross bound} \tag{7}
\]

\[
\sum_{v_i \in V_{\text{zone}}} y_k^i \leq m, \quad k = 1, \ldots, \lceil|V_{\text{zone}}|/m\rceil \tag{8}
\]

\[
\sum_{v_i \in V_{\text{zone}}} x_k^i - m \cdot y_k^i \leq 0, \quad v_i \in V_{\text{zone}}, \quad k = 1, \ldots, \lceil|V_{\text{zone}}|/m\rceil \tag{9}
\]

\(w_{ij}\) in the objective function indicates the number of edges between nodes \(v_i\) and \(v_j\) in the access graph.\(^2\) Constraint (5) states that every variable in \(V_{\text{in}}\) should be assigned to one page, and constraint (6) states that each variable in \(V_{\text{cross}}\) can be assigned to a page, but its assignment is not mandatory and depends on constraint (7), in which \(\text{cross bound}\) is defined as:

\[
\text{cross bound} = \lceil \sum_{v_i \in V_{\text{cross}}} \frac{f_2(v_i)}{f(v_i)} \rceil
\]

where \(f_2(v_i)\) is the number of \(v_i\) accesses in \(\text{access zone} Z\) of access sequence, and \(f(v_i)\) is the number of \(v_i\) accesses in the whole access sequence. We use the value of \(f_2(v_i)/f(v_i)\) as the degree of the possibility of assigning \(v_i\) to a page in the ILP formulation of current \(\text{access zone}\). Constraint (8) ensures the page size constraint. Finally, constraint (9) indicates that \(x_k^i = 1\) for any \(v_i\) implies \(y_k^i = 1\).

From the execution of the ILP-based \(\text{zone alignment}\) for current \(Z\), we update \(V_{\text{in}}\) and \(V_{\text{cross}}\) for the formulation and execution of next \(\text{access zone}\). Note that the gain (i.e., the number of page accesses) for \(Z\) can be computed from the results of the objective function. If

\(^2\)We assume \(w_{ij} = 0\) if \(i = j\) in the access graph simply because the weight of self loops is always included in the gain (in Eq.(1)) regardless of the resultant partition.
we denote the gain at access zone Z by $\text{gain}(Z)$, the total number of page accesses over the whole access sequence is computed by

$$\sum_{i=1}^{T} \text{gain}(Z_i) + \sum_{i=1}^{\infty} w_i^{\text{def}}$$

(10)

where we assumed the input access sequence is partitioned into $T$ access zones $Z_1, Z_2, \ldots, Z_T$, and $w_i^{\text{def}}$ is the number of consecutive accesses ($v_i, v_j$) in the access sequence. (Note that $w_{ij}$ is the weight of edge $(v_i, v_j)$ in access graph except $w_{ij} = 0$ when $i = j$).

If the application of zone alignment to access time interval $[c_{\text{start}}, c_{\text{end}}]$ is done, we determine the next access time interval $[c_{\text{start}}, c_{\text{end}}]$ to be applied where $c_{\text{start}}$ is the very next access time of $c_{\text{end}}$ (i.e., $c_{\text{start}} = c_{\text{end}} + 1$). We repeat this process until all the variables in the access sequence have been assigned to pages. It should be noted that one important issue that affects the quality of our iterative approach is the way of dividing the whole access time interval of access sequence into a set of sub-intervals of manageable size. We control the size of access zone by using a user-defined value $L$ as an upper-limit of the variables in each sub-interval of access sequence. Specifically, we determine the value of end-access time $c_{\text{end}}$ of $[c_{\text{start}}, c_{\text{end}}]$ to the one with minimum value of $|V_{\text{cross}}|$ among the access times that satisfies $L - \alpha \leq |V_{\text{start}}| + |V_{\text{cross}}| \leq L$. (Note that taking the access-time with minimum value of $|V_{\text{cross}}|$ is likely to minimize the degradation of quality of solution incurred from the zone splitting scheme. We use function $\text{GetNextEnd}(S)$ to find such end-access time $c_{\text{end}}$ among access-times in access sequence $S$. Finally, Figure 3 summarizes the overall procedure of the proposed approach zone alignment where, by using different values of $L$, we can trade-off between the quality of solution and run time.

**zone alignment** 

* ILP based page assignment

**Inputs:** Access graph $G(V, E)$, page size $m$, access sequence $S$, user-defined values $L$ and $\alpha$.

**Output:** Page assignment for $V$.

- $Z_{\text{rem}} \leftarrow S$; $* \ Z_{\text{rem}}$: remaining access sequence $* /

- $c_{\text{end}} \leftarrow -1$; $* \text{the access-time before the start of } S \* /

repeat 

{ 
- $c_{\text{start}} = c_{\text{end}} + 1$;
- $c_{\text{end}} \leftarrow \text{GetNextEnd}(Z_{\text{rem}})$;
- Apply the ILP formulation to $[c_{\text{start}}, c_{\text{end}}]$;
- $Z_{\text{rem}} = \lfloor c_{\text{end}} + 1. \text{end of } S \rfloor$; $* \text{update } Z_{\text{rem}} \* /
}

until ($Z_{\text{rem}}$ is empty)

Fig. 3. A summary of the proposed approach zone alignment.

**IV. MEMORY ALIGNMENT FOR MAXIMIZING BURST ACCESES**

**A. Problem Definition**

In this section we consider DRAMs which allow burst mode as well as normal mode (e.g., [5]). The memory layout problem is then to maximize the number of burst accesses. We call it the MLB problem (Memory Layout with Burst mode) of $(S, V, m)$.

**Definition 4.1:** The directed access graph of $S$ is a directed graph $G(V, A)$ where $V$ is the set of variables in $S$ and there are $n$ arcs from $v_i$ to $v_j$ if and only if there are exactly $n$ consecutive references of variables $v_i, v_j$ in $S$ denoting $w(v_i, v_j) = m$.

Let $S = a_1, a_2, \ldots, a_{m-1}, a_1, \ldots, a_{m}$ be a sequence of variable references. Recall that $a_i$ in a memory layout is a burst access if $f_{\text{page}}(v(a_i)) = f_{\text{page}}(v(a_{i-1}))$ and $f_{\text{addr}}(v(a_i)) = f_{\text{addr}}(v(a_{i-1})) + 1$. Thus, if there are $n$ consecutive accesses of variables $v(a_{i-1}), v(a_i)$, and $v(a_{i-1})$ and $v(a_i)$ are stored in consecutive locations of the same page, the $n$ accesses of the $v(a_i)$ will be burst accesses. (See Figure 4(a) for an example of access sequence and its directed access graph.)

**Definition 4.2:** A path cover of $G(V, A)$ is a set of node-disjoint directed paths which collectively cover all the nodes in $V$. A path cover of size $m$ is a path cover $c^{\text{def}}(G(V, A))$ in which every (node-disjoint) path in $c^{\text{def}}(G(V, A))$ covers at most $m$ nodes.

We define a gain of path cover $c^{\text{def}}(G(V, A))$:

$$g(c^{\text{def}}(G(V, A))) = \sum_{(v_i, v_j) \in A, \forall v_i, v_j, \forall c \subseteq c^{\text{def}}(G(V, A))} w_{ij}$$

(11)

**Definition 4.3:** A maximum weighted path cover (MWPC) of size $m$ in $G(V, A)$ is a path cover $c^{\text{def}}(G(V, A))$ that maximizes the quantity in Eq.(11).

The memory layout implied by a path cover $c^{\text{def}}(G(V, A))$ is the one that satisfies: Each path in $c^{\text{def}}(G(V, A))$ is a distinct page in the layout and the sequence of variables on the path is the relative placement order of the variables in the page. For example, Figure 4(c) is the memory layout implied by the path cover in Figure 4(b) when $m = 3$, in which the heavy arrows indicate the path cover.

**IV. PROPOSED TECHNIQUE**

We describe in this section an exact 0-1 ILP formulation of MLB in the context of path cover problem since according to Theorem 4.1, finding an optimal solution of MLB with $(G, V, m)$ is equivalent to finding an optimal path cover solution $c^{\text{def}}(G(V, A))$. Note that the problem of optimal path cover is quite different from the problem of optimal graph partitioning in Section III, in that the paths in a path cover are directed and further the path cover should contain no cycles. Thus, rather than completing our ILP formulation from the one in Section III for the graph partitioning by including additional constraints, we directly formulate the path cover problem into ILP efficiently deal with the directions of the arcs and the prevention of the existence of cycles on the paths.

**Theorem 4.1:** The memory layout implied by optimal MWPC of size $m$ maximizes the number of burst accesses.

**Theorem 4.2:** The MLB problem is NP-hard.

**B. The Proposed Technique**

Fig. 4. (a) An access sequence and its directed access graph; (b) a path cover; (c) the memory layout implied by the path cover in (b).

Our ILP formulation is derived based on the following two theorems from the work in [11]:

**Theorem 4.1:** The memory layout implied by optimal MWPC of size $m$, maximizes the number of burst accesses.

**Theorem 4.2:** The MLB problem is NP-hard.

We describe in this section an exact 0-1 ILP formulation of MLB in the context of path cover problem since according to Theorem 4.1, finding an optimal solution of MLB with $(G, V, m)$ is equivalent to finding an optimal path cover solution $c^{\text{def}}(G(V, A))$. Note that the problem of optimal path cover is quite different from the problem of optimal graph partitioning in Sec. III, in that the paths in a path cover are directed and further the path cover should contain no cycles. Thus, rather than completing our ILP formulation from the one in Sec. III for the graph partitioning by including additional constraints, we directly formulate the path cover problem into ILP to efficiently deal with the directions of the arcs and the prevention of the existence of cycles on the paths.

**Theorem 4.1:** The memory layout implied by optimal MWPC of size $m$, maximizes the number of burst accesses.

**Theorem 4.2:** The MLB problem is NP-hard.

We describe in this section an exact 0-1 ILP formulation of MLB in the context of path cover problem since according to Theorem 4.1, finding an optimal solution of MLB with $(G, V, m)$ is equivalent to finding an optimal path cover solution $c^{\text{def}}(G(V, A))$. Note that the problem of optimal path cover is quite different from the problem of optimal graph partitioning in Sec. III, in that the paths in a path cover are directed and further the path cover should contain no cycles. Thus, rather than completing our ILP formulation from the one in Sec. III for the graph partitioning by including additional constraints, we directly formulate the path cover problem into ILP to efficiently deal with the directions of the arcs and the prevention of the existence of cycles on the paths.

Given $(S, V, m)$, let us consider a graph of $m$ columns of variables as shown in Figure 5 where each column is arranged by $n = |V|$ variables $v_1, \ldots, v_n \in V$. Note that we can easily check that the paths in any path cover for a directed access graph can be represented as disjoint paths in the corresponding graph of $m$ columns of variables. Then, the problem is to find a set of disjoint paths of size $m$ or less from $m$ the $m$-columned graph corresponding to access graph such that the total number of nodes in the paths should be exactly $n = |V|$ in the access graph) and some objective (defined later) be maximized. Note that since we create arcs only between the nodes in two adjacent columns of the graph as in Figure 5, the length of the longest (directed) path in the graph never exceeds $m$. To describe the disjoint paths, we introduce a new 0-1 valued variable $x_{ij}$ for two variables $v_i$ in the $k$th column and $v_j$ in the $(k + 1)$th column as shown in Figure 5 and associate it with

$$x_{ij} = \begin{cases} 1, & \text{if arc from } v_i \text{ to } v_j \text{ is on a disjoint path,} \\ 0, & \text{if arc from } v_i \text{ to } v_j \text{ is not on any disjoint path.} \end{cases}$$

(12)

The length of path is defined to be the number of nodes on the path.
Then, the ILP formulation using the \( x_{ji} \) variables are:

\[
\sum_{i=1}^{[V]} \sum_{j=1}^{[V]} \sum_{k=1}^{m} w_{ij} x_{ji} = 1, \quad j = 1, \ldots, [V]
\]

subject to,

\[
\sum_{j=1}^{[V]} x_{ji} + \sum_{i=1}^{[V]} x_{ji} \leq 1, \quad p = 2, \ldots, m, \quad j = 1, \ldots, [V]
\]

\[
\sum_{i=1}^{[V]} x_{pi} + \sum_{i=1}^{[V]} x_{pi} \leq 1, \quad p = 1, \ldots, m - 1, \quad \forall j = 1, \ldots, [V]
\]

Constraints (14) and (15) ensure that if a disjoint path passes through node \( v_i \) in the \( p \)th column, the path cannot go through node \( v_j \) in each of the other columns. We found that two constraints are enough to find an optimal solution of MWPC.

If the problem size is large, we use zone_alignment technique for MLB by including addition constraints to the ILP formulation described above. We introduce another 0-1 valued variable \( y_i \) to associate it with node \( v_i \in V \), \( i = 1, \ldots, [V] \):

\[
y_i = \begin{cases} 1 & \text{if node } v_i \text{ in some column is covered by a disjoint path, } \\
0 & \text{otherwise.} 
\end{cases}
\]

Then we have new ILP formulation for zone_alignment technique:

\[
\sum_{i=1}^{[V]} \sum_{j=1}^{[V]} \sum_{k=1}^{m} w_{ij} x_{ji} = 1
\]

subject to,

\[
\sum_{j=1}^{[V]} x_{ji} + \sum_{i=1}^{[V]} x_{ji} \leq 1, \quad p = 2, \ldots, m, \quad j = 1, \ldots, [V]
\]

\[
\sum_{i=1}^{[V]} x_{pi} + \sum_{i=1}^{[V]} x_{pi} \leq 1, \quad p = 1, \ldots, m - 1, \quad \forall j = 1, \ldots, [V]
\]

\[
\sum_{k=1}^{[V]} \sum_{i=1}^{[V]} x_{kij} + \sum_{i=1}^{[V]} x_{kij} \leq 2 - 2y_i \leq 0, \quad j = 1, \ldots, [V]
\]

\[
y_i \leq |V| + \text{cross bound}
\]

Constraints (17) and (18) are the constraint (14) and (15), respectively. Constraint (19) states that if node \( v_j \) in some column is covered by a path, (not a single node itself), \( y_j \) is set to 1, and constraint (20) states that the number of variables that are to be assigned to pages should be no more than \( |V| + \text{cross bound} \).

V. EXPERIMENTAL RESULTS

We conducted a set of experiments to check the effectiveness of the proposed algorithm. We used cplex as the ILP solver on Linux machine with Intel Xeon 3 500 X4. We tested our algorithm on a set of benchmark programs in [12], [13], [14] (See Table III), and compared our results with that produced by the OFU (the order of first use) variable assignment, the CGB (the closeness-graph-based) algorithm [1,2], and the Greedy heuristic [11].

VI. CONCLUSIONS

In this paper, we provided a set of important/interesting results for the storage assignment problems (which is known to be intractable) of variables with an objective of maximizing the number of page/burst mode accesses: We solved the problem by proposing an ILP-based near-optimal technique, called zone_alignment, utilizing the temporal locality of variable accesses in code.

REFERENCES