# Memory Minimization for Tensor Contractions using Integer Linear Programming

A. Allam[1], J. Ramanujam[1], G. Baumgartner[2], and P. Sadayappan[3]

[1] Department of Electrical and Computer Engineering, Louisiana State University, USA
[2] Department of Computer Science, Louisiana State University, USA
[3] Department of Computer Science and Engineering, The Ohio State University, USA
{atef,jxr}@ece.lsu.edu, {gb,saday}@cse.ohio-state.edu

## Abstract

This paper presents a technique for memory optimization for a class of computations that arises in the field of correlated electronic structure methods such as coupled cluster and configuration interaction methods in quantum chemistry. In this class of computations, loop computations perform a multi-dimensional sum of product of input arrays. There are many different ways to get the same final results that differ in the required number of arithmetic operations required. In addition, for a given number of arithmetic operations, different expressions of the loop have different memory requirements. Loop fusion is a plausible solution for reducing memory usage. By fusing loops between producer loop nest and consumer loop nest, the required storage of intermediate array is reduced by the range of the fused loop. Because resultant loops have to be legal after fusion, some loops can not be fused at the same time. In this paper, we have developed a novel integer linear programming (ILP) formulation that is shown to be highly effective on a number of test cases producing the optimal solutions using very small execution times. The main idea in the ILP formulation is the encoding of legality rules for loop fusion of a special class of loops using logical constraints over binary decision variables and a highly effective approximation of memory usage.

## 1. Introduction

The class of computations considered in this work arises in the field of correlated electronic structure methods such as coupled cluster and configuration interaction methods in quantum chemistry [Aul96, Hyb86, Roj95]. In this class of computations, loop computations are specified as multi-dimensional integrals of products of many input arrays. These computations can be expressed numerically as multi-dimensional sums of products of input arrays. There are many different ways to get the same final results; the different ways require differing number of arithmetic operations, due to operator properties such as commutativity, associativity and distributivity. Lam et al. [Lam97] have devised an optimization procedure to do loop computations using the minimum number of floating point operations through determining an equivalent sequence of multiplication and summation formulas; the resulting optimal sequence of formulas is called an *operation-count-optimal* formula sequence. The intermediate result from each formula is stored in an intermediate array that can be used many times without the need for re-computing these results.

Resulting formulas can be implemented as separate sets of perfectly nested loops, one set for each formula. In this way, intermediate arrays have to be stored in full; in most cases they are huge and often exceed the available memory on most machines. Loop fusion [Gao92, Ken93, Lam97, Lam99a, Lam02, Man95, Sin96] is a candidate solution for reducing memory usage. By fusing loops between producer loop nest and consumer loop nest, the required storage of intermediate arrays is reduced by the range of the fused loop. Because resultant loops have to be legal after fusion, fusing some loops precludes fusing others. The problem of deciding which loops are to be fused to achieve minimal memory usage is called the *optimal memory usage* problem, which is what we are considering in this work.

## 2. Problem Definition and Formulation

Figure 1 is an example of a multi-dimensional integral; Figure 1(a) shows a multi-dimensional integral expressed as a sum of product of arrays. Figure 1(b) shows the resultant operation-count-optimal formula sequence and Figure 1(c) is its graph representation. This graph representation is the same as the one presented in [Lam02] except that the multiplication and summation nodes are combined together in one node.

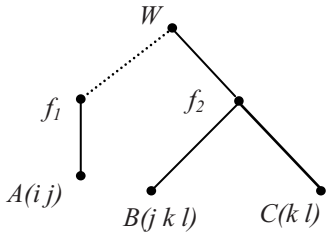
1-4244-0054-6/06/$20.00 

$$W[k] = \sum_{i,j,l} A[i,j] \times B[j, k, l] \times C[k, l]$$

**(a)**

$$f_1[j] = \sum_i A[i,j]$$
$$f_2[j, k] = \sum_l B[j, k, l] \times C[k, l]$$
$$W[k] = \sum_j f_1[j] \times f_2[j, k]$$

**(b)**




**(c)**

Figure 1: An example of multi-dimensional integral. **(a)** A multi-dimensional integral; **(b)** A formula sequence for computing (a); **(c)** Graph representation of (b)

## 2.1 Modified Fusion Graph

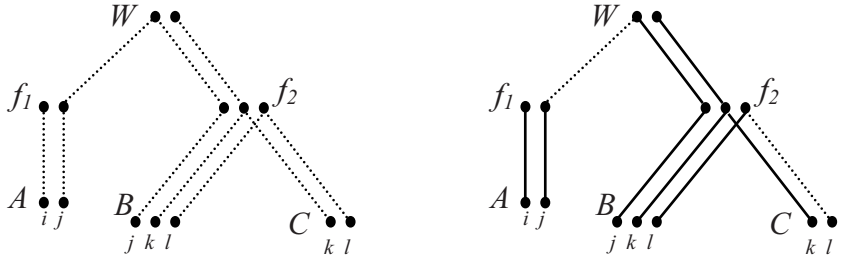
Another graph representation of the problem called the *fusion graph FG = (V, E, Inds, N)* is extracted from the original problem graph, which is suitable for formulating the fusion problem at hand and it is a modified version of the one presented in [Lam02] in order to decrease the number of variables used. In the fusion graph FG,

- *V* is the set of nodes where each node represents an array (intermediate, input, or output array),
- *E* is the set of potential fusion edges as described below,
- *Inds* are the sets of loop indices associated with each node, and
- *N* is the set of loop ranges.

The fusion graph is constructed as follows:

- Each node $v \in V$ in the original graph is converted to a set of vertices, one for each loop *i*, where *i* is a loop index of node *v* ($i \in Inds(v)$).
- For each common loop index between a node and its parent, an edge $e \in E$ is introduced called a *potential fusion edge*; the common loop index in this case is said to be a candidate for fusion. If the common loop between a node and its parent is fused, the potential fusion edge is called a *fusion edge*.

Figure 2(a) shows the potential fusion graph for the original graph in Figure 1(c). The potential fusion edges are dotted edges and the fusion edges are shown as solid edges in a fusion graph. In a fusion graph, each connected component of fusion edges forms a fusion chain, which corresponds to a fused loop in the loop structure. In Figure 2(b), there are three fusion chains, one for each of the *j-, k-,* and the *l*-loops. The set of nodes between and including nodes *a* and *b*, in which all the *i*-vertices of these nodes are connected through potential fusion edges is called the potential fusion scope of an *i*-loop between the two nodes, *a* and *b*, written as *pfscope(a ,b, i)*. Similarly, the fusion scope of the *i*-loop between two nodes, *a* and *b*, written as *fscope(a ,b, i)*, is defined as the set of nodes between and including *a* and *b*, in which all the *i*-vertices of these nodes are connected through fusion edges. Again, in Figure 2(b), *fscope(B, W, j)* = <*B, f2, W*>.




**(a)** **(b)**

Figure 2: Fusion graph for operation-minimal sequence in Figure 1.
**(a)** Potential fusion graph; **(b)** Resulting fusion graph

Lam et al. [Lam 02] describe the effect of fusion as follows: "In general, fusing a t-loop between a node v and its parent eliminates the t-dimension of the array v and reduces the array size by a factor of Nt. In other words, the size of an array after loop fusions equals the product of the ranges of the loops that are not fused with its parent. We only consider fusions of loops among nodes that are all transitively related by (i.e., form a transitive closure over) parent-child relations. Fusing loops between unrelated nodes (such as fusing siblings without fusing their parent) has no effect on array sizes. We also restrict our attention to loop fusion configurations that do not increase the operation count. In the class of loops considered, the only dependence relations are those between children and parents, and array subscripts are simply loop index variables. Loop permutations, loop nests reordering, and loop fusions are, therefore, always legal as long as child nodes are evaluated before their parents. This freedom allows the loops to be permuted, reordered, and fused in a large number of ways that differ in memory usage." [Lam02].

We have proposed a mathematical formulation for the optimal memory usage problem in which the objective is to minimize the total memory usage (static memory allocation model) for the given operation-count-optimal formula sequence. Constraints to assure legality for the resultant fusion graph are developed in a form of a set of linear inequalities. Because of the nature of the problem, the objective is formulated as a nonlinear function. Then, an efficient linearization technique has been developed that transforms the objective function to be linear and thus the memory usage problem is formulated as an integer linear programming (ILP) problem. Although the linearized objective function does not guarantee optimality, the solution is found to match the optimal one in several cases because the linearization we have devised is an effective approximation of the nonlinear objective function.

## 3. Legality of Fusion

The following theorem states the basic definitions and the sufficient conditions for a fusion to be legal. And based on that theorem, fusion legality constraints are generated.

**Theorem 1:** Let *FG = (V, E, Inds, N*) be a fusion graph, and let *a* and *b* be any two nodes in FG. For any two loop-indices *j* and *k*, ***fusion is legal if one of the following conditions is satisfied:***

1. $fscope(a, b, j) \cap fscope(a, b, k) = \varnothing$.
2. $fscope(a, b, j) \subseteq fscope(a, b, k)$.
3. $fscope(a, b, j) \supseteq fscope(a, b, k)$.

**Proof:** Since loops are not allowed to overlap (they must either be nested or separate), fusion is legal if the chains of any two loops in a fusion graph are not partially overlapped, i.e., they must be either disjoint or a subset/superset of each other, which can be mathematically rewritten as the conditions (1)-(3) above. ■

Figure 3 shows different cases of illegal fusion and Figure 4 shows different configurations of legal fusion.

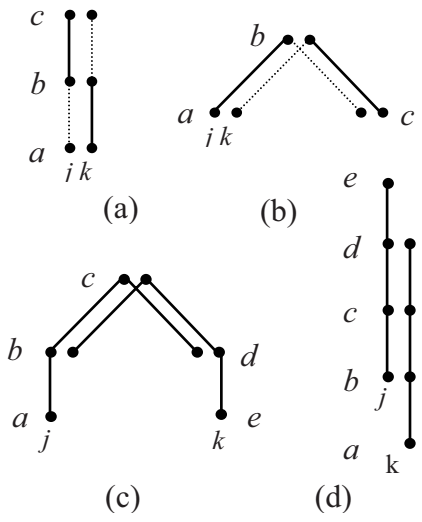



Figure 3: Illegal Fusion configurations.

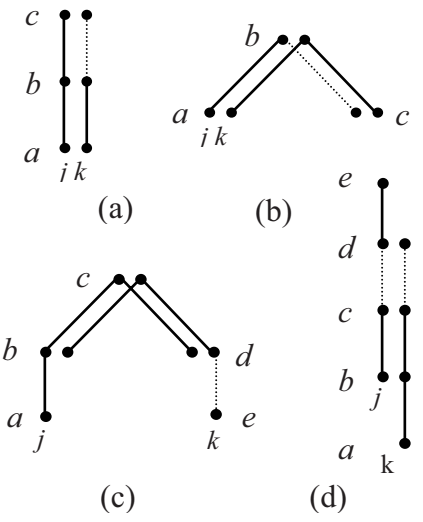



Figure 4: Legal Fusion configurations.

*for each path* $P(s,t)$ *and*
*for each candidate pair of loop indices* $j$ *and* $k$ :

$$(x_{sj}+x_{tk})-(x_{sk}+x_{tj}) \le 1+\sum_{a\in P(s,t)-\{s,t\}}(1-x_{ak})$$

and $$(x_{sk}+x_{tj})-(x_{sj}+x_{tk}) \le 1+\sum_{a\in P(s,t)-\{s,t\}}(1-x_{aj}) \quad (1)$$

which can be rewritten as:

$$(x_{sj}-x_{tj})+\left(-x_{sk}+x_{kj}+\sum_{a\in P(s,t)-\{s,t\}}x_{ak}\right)\le 1+m$$

*and* $$(x_{sk}-x_{tk})+\left(-x_{sj}+x_{tj}+\sum_{a\in P(s,t)-\{s,t\}}x_{aj}\right)\le 1+m \quad (2)$$

where $m$ is the number of intermediate nodes in the path $P(s,t)$.

To capture the legality of fusion in a set of linear inequalities, we introduce a *0-1* unknown variable, $x_{ai}$, to denote the fusion edge between node *a* and its parent. The unknown variable $x_{ai}$ takes a value 1 if the *i-loop* is fused between node *a* and its parent, and 0 otherwise.

Fusion legality described by Theorem 1 can be posed as constraints in form of linear inequalities using Equation (1) shown below. That simply says: for each path *P(s,t)* that starts at node *s* and ends at node *t*, and for any two loop indices *j* and *k* in the fusion graph, a constraint in the form of Equation (1) is generated as long as

1. both of these two loops are candidates for fusion (i.e., there are potential fusion edges between each intermediate node that belongs to that path and its parent for both *j-* and *k-loop*), and
2. at least one potential fusion edge for node *s* and node *t* where the two loop indices are different, i.e., one node has potential fusion edge for the *j-loop* and the other for the *k-loop*.

Although the first term in the right-hand side of Equations (1) and (2) is enough to guarantee legality for most of the fusion configurations, the second term is needed to take care of some legal configurations such as the one in Figure 4(d); without the second term, the configuration in Figure 4(d) appears to be illegal even though it is legal.

A depth-first search algorithm is used for path construction for generating fusion legality constraints. The fusion graph is treated as an undirected acyclic graph during path traversal, i.e., the notion of parent or child is no longer considered during path traversal. At the same time, the fusion edge definition is still as it is in the original graph. For example, consider the fusion graph shown in Figure 2(d) where node *c* is the parent of both node *b* and node *d*. In constructing the path *P(b,e)* = <*b, c, d, e*> that originates at node *b* and ends at node *e* for loop indices *j* and *k*, the unknown variable corresponding to the candidate fusion edge *(e,d)* is $x_{ek}$.

# 4. ILP formulation

## 4.1 Fusion Constraints

The fusion legality constraints in inequalities (1) and (2) work as the set of constraints in the ILP formulations; the objective function developed below completes the formulation. The number of fusion legality constraints may appear to be large, but in practice our experience with several benchmark expressions from computational chemistry indicates that most of the candidate pairs of loop indices do not exist in all nodes in the fusion graphs; this renders constraint (1) inapplicable to most of the paths and hence these constraints are not generated in the ILP formulation. Moreover, as shown in inequality (1), the coefficients of the constraint matrices are 1's or 0's, which plays a substantial role in decreasing the solution time from the ILP models as demonstrated in our experimental results.

## 4.2 Fusion Objective Function

Since the objective is to minimize memory usage, an expression for memory usage of an array needs to be developed. Equation (3) shows the memory usage for a multidimensional array 'A' assuming that the fused loops and unfused loops are known. The memory requirement for array 'A' is the product of the sizes along the unfused dimensions of array 'A'. Using the associated 0-1 variables introduced in the ILP formulation, in the expression as a trial to get a mathematical formula eligible to be used in an objective function, Equation (4) results. Summing over all the arrays (nodes in the fusion graph), the total-memory usage can be

used as an objective function as shown in Equation (6).

$$mem(A) = \prod_{\substack{i \in Inds(A) \\ and\ i\ is unfused}} N_i \quad (3)$$

$$mem(A) = \prod_{\substack{i \in Inds(A) \\ and\ i\ is unfused}} (1 - x_{Ai}) N_i \quad (4)$$

*Minimize* :

$$total\ memory = \sum_A \left[ \prod_{\substack{i \in Inds(A) \\ and\ i\ is unfused}} (1 - x_{Ai}) N_i \right] \quad (5)$$

Equation (5) is not a suitable form for a mathematical formulation to express an objective function to be minimized. This is because the unfused loops are not known apriori to restrict the memory expression to include only the unfused loops. Also, taking off the restriction and including all the loop indices in the memory expression creates another problem, in that only one loop to be fused in a multi-dimensional array is enough to make the memory contribution of this array in the objective function to be zero. In this way, the effect of fusing one loop in a multi-dimensional array has the same effect as fusing two or more loops, which is not the optimal solution. For example, consider a three-dimensional array A with loop indices *i*, *j*, and *k*; mem(A) = $(1-x_{Ai})(1-x_{Aj})(1-x_{Ak})N_iN_jN_k$. Fusing only loop i results in the same objective function value as fusing loops *i* and *j*. But in the first case mem(A) = $N_jN_k$, and in the second case mem(A) = $N_k$.

The accurate memory expression of a multi-dimensional array should include the different combinations of resulting memory after fusion including all its loops as shown in Equation (6). Consider again the three-dimensional array A above, its memory expression is as in Equation (7).

$$\begin{aligned} mem(A) = {} & resultant\ memory\ of\ array\ A \\ & if\ none\ of\ the\ loops\ are\ fused. \\ & + if\ one\ loop\ is\ fused\ at\ a\ time. \\ & + if\ two\ loops\ are\ fused\ at\ a\ time \\ & + \ \ldots \\ & + if\ all\ loops\ are\ fused. \end{aligned} \quad (6)$$

$$\begin{aligned} mem(A) = {} & (1- x_{Ai})(1-x_{Aj})(1-x_{Ak})N_iN_jN_k + \\ & (x_{Ai})(1-x_{Aj})(1-x_{Ak})N_jN_k + (1-x_{Ai})(x_{Aj})(1-x_{Ak})N_iN_k \\ & + (1-x_{Ai})(1-x_{Aj})(x_{Ak})N_iN_j + x_{Ai}x_{Aj}(1-x_{Ak})N_k \\ & + x_{Ai}(1-x_{Aj})x_{Ak}N_j + (1- x_{Ai})x_{Aj}x_{Ak}N_i + x_{Ai}x_{Aj}x_{Ak} \end{aligned} \quad (7)$$

Using the memory expression in (6) in the objective function results in a nonlinear objective function that needs a nonlinear solver which is expensive and inefficient (in terms of solution time). Thus we resort to linearization.

### 4.3 Objective Function Linearization

A direct way to linearize Equation (6) is by summing over all the complements of the *0-1* variables $x_{Ai}$'s weighted by the corresponding loop-ranges $N_i$'s, as shown in Equation (8). Since the objective is minimization, a maximum number of loops are fused as long as the fusion legality is satisfied giving more preference to the loops with larger dimensions.

$$\text{Minimize: } \sum_A \sum_{i \in Inds(A)} (1 - x_{Ai}) N_i \quad (8)$$

This can be rewritten as:

$$\text{Maximize: } \sum_A \sum_{i \in Inds(A)} x_{Ai} N_i \ . \quad (9)$$

Linearization as defined in Equation (9) is exact only when all the arrays are one-dimensional arrays but this is not the general case. For example, consider a two-dimensional array A, and loop indices *i* and *j* in A with loop-ranges 10 and 15 respectively, and a one-dimensional array B that has loop index *k* with loop-range 20 and assume that the solver has to choose between *j*- and *k*-loops to fuse because of legality constraints. Applying Equation (9), the objective function $f_{obj}$ will be: $f_{obj} = 10x_{Ai} + 15x_{Aj} + 20x_{Bk}$. Because the objective in (9) is a maximization problem, the ILP solver will set $x_{Bk}$ to 1 and $x_{Aj}$ to 0, which results in $f_{obj}$ = 30 and the total memory for this case is 150 + 1 = 151. On the other hand, if had set $x_{Aj}$ to 1 and $x_{Bk}$ to 0, $f_{obj}$ = 15 (which is less than the other case), but this will result in an optimal memory usage with total memory = 10 + 20 = 30. This is the key idea used in the efficient linearization of the objective function given by the following function

$$\text{Maximize: } \sum_A \sum_{i \in Inds(A)} x_{Ai} \left( size(A) - rsize(A,i) \right) \quad (10)$$

where *size(A)* is the memory size of array *A* and *rsize(A,i)* is the reduced memory size of array *A* if the *i*-loop is fused. The expressions for these turn out to be easily expressed as

$$size(A) = \prod_{i \in Inds(A)} N_i, \quad rsize(A,i) = \prod_{\substack{k \in Inds(A) \\ and\ k \neq i}} N_k$$

The expression *size(A) – rsize(A,i)* expresses the reduction in memory for array *A* if the *i*-loop is fused between node *A* and its parent. From the previous example, *size(A) - rsize(A,i)* = 150 – 15 =

135, *size(A) - rsize(A,j)* = 150 – 10 = 140, and *size(B) - rsize(B,k)* = 20 – 1 = 19. On plugging these values in Equation (10), we get $f_{obj} = 135x_{Ai} + 140x_{Aj} + 19x_{Bk}$. Because the objective is maximization, the ILP solver will pick $x_{Aj}$ to be one and $x_{Bk}$ to be zero, this will result in an optimal memory usage for this example with total memory = 10 + 20 = 30.

## 5. Example

Consider the potential fusion graph in Figure 2(a) (assuming that the ranges for loops *i, j, k, l* are 10, 10, 12, and 10 respectively). The associated 0-1 variables for each potential fusion edge is shown in Table 1; the complete ILP formulation is as shown in Figure 5 below. The associate path and its pair of loop indices for each set of constraints in Figure 5 are as shown in Table 2. The output of the ILP solver is shown in Figure 2(b), where the solid lines represent the resulting fused edges.

## 6. Experimental Results

We have tested our ILP formulation on test examples taken from [Lam02] that arise in the field of correlated electronic structure methods such as coupled cluster and configuration interaction methods in quantum chemistry. Table 3 shows the comparison between the memory usage results from the optimal solution and our ILP formulation. It also shows that our formulation is efficient where the solution time is a fraction of seconds even for large test cases. Additional details on the benchmarks can be found in the first author's PhD thesis [All05].

Table 1: Potential fusion edges and their associated 0-1 variables

| edge | (A,$f_1$) | (A,$f_1$) | ($f_1$,W) | (B,$f_2$) | (B,$f_2$) | (B,$f_2$) | (C,$f_2$) | (C,$f_2$) | ($f_2$,W) | ($f_2$,W) |
|---|---|---|---|---|---|---|---|---|---|---|
| loop-index | i | j | j | j | k | l | k | l | j | k |
| variable | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | x10 |

```
Maximize:   obj: 90 x1 + 90 x2 + 9 x3 + 1080 x4 + 1100 x5+
                 1080 x6 + 110 x7 + 108 x8 + 108 x9 + 110 x10
Subject To:
 c1:  x1 - x2 + x3 <= 1
 c2:  - x1 + x2 - x3 <= 1
 c3:  x4 - x5 - x9 + x10 <= 1
 c4:  - x4 + x5 + x9 - x10 <= 1
 c5:  - x7 - x9 + x10 <= 1
 c6:  x7 + x9 - x10 <= 1
 c7:  x4 - x6 - x9 <= 1
 c8:  - x4 + x6 + x9 <= 1
 c9:  - x8 - x9 <= 1
 c10: x8 + x9 <= 1
 c11: x5 - x6 - x10 <= 1
 c12: - x5 + x6 + x10 <= 1
 c13: x7 - x8 - x10 <= 1
 c14: - x7 + x8 + x10 <= 1
 c15: x4 - x5 + x7 <= 1
 c16: - x4 + x5 - x7 <= 1
 c17: x3 - x9 + x10 <= 1
 c18: - x3 + x9 - x10 <= 1
 c19: x3 - x4 + x5 + x10 <= 2
 c20: - x3 + x4 - x5 + x9 <= 2
 c21: x3 + x7 + x10 <= 2
 c22: - x3 - x7 + x9 <= 2
 c23: x4 - x6 + x8 <= 1
 c24: - x4 + x6 - x8 <= 1
 c25: x5 - x6 - x7 + x8 <= 1
 c26: - x5 + x6 + x7 - x8 <= 1
```

Figure 5: Complete ILP formulation for the fusion graph in Figure 2(a)

Table 2: Associated paths and their pairs of loop indices for the constraints in Figure 5

| constraints | associated path | loop indices |
|---|---|---|
| c1, c2 | < A, $f_1$, W> | *i, j* |
| c3, c3 | <B, $f_2$, W> | *j, k* |
| c5, c6 | <C, $f_2$, W> | *j, k* |
| c7, c8 | <B, $f_2$, W> | *j, l* |
| c9, c10 | <C, $f_2$, W> | *j, l* |
| c11, c12 | <B, $f_2$, W> | *k, l* |
| c13, c14 | <C, $f_2$, W> | *k, l* |
| c15, c16 | <B, $f_2$, C> | *j, k* |
| c17, c18 | < $f_1$, W, $f_2$> | *j, k* |
| c19, c20 | < $f_1$, W, $f_2$, B> | *j, k* |
| c21, c22 | < $f_1$, W, $f_2$, C> | *j, k* |
| c23, c24 | <B, $f_2$, C> | *j, l* |
| c25, c26 | <B, $f_2$, C> | *k, l* |

Table 3: Fusion Results on Benchmarks

| Problem | Optimal Sol | Our ILP | time(sec) | |
|---|---|---|---|---|
| Test 0 | 23 | 23 | 0.01 | |
| Test 1 | 2.700909300006 e+12 | 2.700909300006 e+12 | 0.05 | |
| Test 2 | 2.700900000206000 e+12 | 2.700900000206000 e+12 | 0.17 | |
| Test 3 | 600008 | 600008 | 0.10 | |
| Test 4 | 1.809003105 e+9 | 1.809003105 e+9 | 0.09 | |
| Test 5 | 1.62027018006003 e+14 | 1.62027018006003 e+14 | 0.18 | |

## 7. Conclusions

This paper presented a technique for memory optimization for a class of computations that arises in the field of correlated electronic structure methods such as coupled cluster and configuration interaction methods in quantum chemistry. In this class of computations, loop computations perform a multi-dimensional sum of product of input arrays. There are many different ways to get the same final results that differ in the required number of arithmetic operations required. In addition, for a given number of arithmetic operations, different expressions of the loop have different memory requirements. Loop fusion is a plausible solution for reducing memory usage. By fusing loops between producer loop nest and consumer loop nest, the required storage of intermediate array is reduced by the range of the fused loop. Because resultant loops have to be legal after fusion, some loops can not be fused at the same time. In this paper, we have developed a novel integer linear programming (ILP) formulation that is shown to be highly effective on a number of test cases producing the optimal solutions using very small execution times. The main idea in the ILP formulation is the encoding of legality rules for loop fusion of a special class of loops using logical constraints over binary decision variables and a highly effective approximation of memory usage. Work is in progress in incorporating different objective functions to precisely capture memory usage. In addition, we plan to explore ways to incorporate disk access costs.

## Acknowledgments

We gratefully acknowledge the support provided in part by the US National Science Foundation through awards CHE-0121676, CHE-0121706, CCF-0508245, CNS-0509442, and CNS-0509467.

## References


[All05] A. Allam, *Power and Memory Optimization Techniques in Embedded Systems*. Ph.D. Dissertation, Louisiana State University, Baton Rouge, LA, August 2005.

[Aul96] W. Aulbur, *Parallel implementation of quasi-particle calculations of semiconductors and insulators*. Ph.D. Dissertation, The Ohio State University, Columbus, October 1996.

[Cha93] S. Chatterjee, J. R. Gilbert, R. Schreiber, and S.-H. Teng, "Automatic array alignment in data-parallel programs," In *Proc. 20th Annual ACM SIGACT/SIGPLAN Symposium on Principles of Programming Languages*, New York, pp. 16–28, 1993.

[Cha95] S. Chatterjee, J. R. Gilbert, R. Schreiber, and S.-H. Teng, "Optimal evaluation of array expressions on massively parallel machines," *ACM TOPLAS*, 17 (1), pp. 123–156, Jan. 1995.

[Fis91] C. N. Fischer and R. J. LeBlanc Jr., *Crafting a compiler*, Benjamin/Cummings, Menlo Park, CA, 1991.

[Gui78] L. J. Guibas and D. K. Wyatt, "Compilation and Delayed Evaluation in APL," *Fifth Annual ACM Symposium on Principles of Programming Languages*, Tucson, Arizona, pp. 1–8, Jan. 1978.

[Gao92] G. Gao, R. Olsen, V. Sarkar, and R. Thekkath, "Collective loop fusion for array contraction," *Languages and Compilers for Parallel Computing*, New Haven, CT, August 1992.

[Hyb86] M. S. Hybertsen and S. G. Louie, "Electronic correlation in semiconductors and insulators: band gaps and quasiparticle energies," Phys. Rev. B, 34 (1986), pp. 5390.

[Ken93] K. Kennedy and K. S. McKinley, "Maximizing loop parallelism and improving data locality via loop fusion and distribution," *Languages and Compilers for Parallel Computing,* Portland, OR, pp. 301–320, August 1993.

[Lam99] C. Lam, D. Cociorva, G. Baumgartner, and P. Sadayappan, "Memory-optimal evaluation of expression trees involving large objects," Technical Report OSU-CISRC-5/99-TR13, Dept. of Computer and Information Science, The Ohio State University, May 1999.

[Lam97] C. Lam, P. Sadayappan, and R. Wenger, "On optimizing a class of multi-dimensional loops with reductions for parallel execution," *Parallel Processing Letters*, Vol. 7 No. 2, pp. 157–168, 1997.

[Lam97a]C. Lam, P. Sadayappan, and R. Wenger, "Optimization of a class of multi-dimensional integrals on parallel machines," Eighth SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis, MN, March 1997.

[Lam99] C. Lam, P. Sadayappan, D. Cociorva, M. Alouani, and J. Wilkins, "Performance optimization of a class of loops involving sums of products of sparse arrays," Ninth SIAM Conference on Parallel Processing for Scientific Computing, San Antonio, TX, March 1999.

[Lam99a] C. Lam, *Performance optimization of a class of loops implementing multi-dimensional integrals*, PhD thesis, Technical Report OSU-CISRC-8/99-TR22, Dept. of Computer and Information Science, The Ohio State University, Columbus, August 1999.

[Lam02] C. Lam, G. Baumgartner, D. Cociorva, and P. Sadayappan, "Memory Minimization for a Class of Loops Implementing Multi-Dimensional Integrals" manuscript, 2002.

[Man95] N. Manjikian and T. S. Abdelrahman, "Fusion of Loops for Parallelism and Locality," *International Conference on Parallel Processing*, pp. II:19–28, Oconomowoc, WI, August 1995.

[Roj95] H. N. Rojas, R. W. Godby, and R. J. Needs, "Space-time method for Ab-initio calculations of self-energies and dielectric response functions of solids," *Phys. Rev. Lett.*, 74 (1995), pp. 1827.

[Sin96] S. Singhai and K. McKinley, "Loop Fusion for Data Locality and Parallelism," *Mid-Atlantic Student Workshop on Programming Languages and Systems*, SUNY at New Paltz, April 1996.