Dynamic Memory Usage Optimization using ILP

A. Allam[1] and J. Ramanujam[2]

[1]Electrical Engineering Dept., Assiut University, Egypt
[2]Electrical and Computer Engineering Dept, Louisiana State University, USA
atef @aun.edu.eg, jxr@ece.lsu.edu

Abstract

In this paper, we address the problem of memory usage optimization in the evaluation of data-dependent program regions involving large data objects. We are interested in situations where the data objects are so large to fit in memory that they have to be dynamically allocated and deallocated during expression evaluation. This problem arises in the scientific computing field such as electronic structure calculations, and in several other contexts. We are considering three different variations of the memory usage optimization problem. The first problem is to find an evaluation order for the dependent data objects to achieve the least amount of memory required. The second problem is the generalization of the evaluation order problem in which the data objects have different processing times and the least amount of memory is sought for a given total execution time as a constraint. The third problem is complement of the second problem in that the total execution time needs to be minimized for a given constraint in memory size. We develop an ILP formulation to optimally solve the three problems.

I. Introduction

We address a general form of the memory usage optimization problem in evaluating given computation represented as a data-flow graph (DFG) *G(V, E, M)* whose nodes *V* represent large data objects of different sizes *M* like those arising in scientific computing such as electronic structure calculations [1, 2, 3] (an expression tree is a special case of the problem we are addressing). Edges *E* in the DFG represent dependencies between these data objects where the evaluation of a data object cannot start until all its children are evaluated. Each data object has to be allocated a certain amount of memory before it can start to be evaluated and it needs to be kept in memory until all its parents are evaluated completely. It is also assumed that each data object is an integral entity that has to be allocated or deallocated as a whole in memory. Reserving memory space for all data objects at the same time requires huge amount of memory and in most cases the available memory is inadequate. The dynamic memory allocation model (in which a data object is allocated memory when it is needed and lasting until all its parents are evaluated and then it is deallocated) is considered in solving this problem.

There are many different possibilities for the evaluation order of the DFG nodes (the large data objects) varying widely in the maximum memory usage. The problem is to find an evaluation order for these data objects to achieve the least memory usage. Two variations of data objects evaluation order problem in the case of multiple-processors using shared memory are also considered. These problems are the performance-constrained evaluation (PCE) and memory-constrained evaluation (MCE) given below. Performance-constrained evaluation addresses the problem of finding an evaluation order for the input DFG nodes that achieves the least amount of memory space required to do the computations without violating the input total execution time (assuming that the execution time required by each node is given). While the memory-constrained evaluation problem dealing with the reverse scenario; where the evaluation order is sought to achieve the minimum total execution time for the DFG under the given memory constraint.

A related work is presented in [4, 5] but it deals with the register allocation problem. The problem of finding an evaluation order of nodes in a given expression tree that uses the least amount of memory is addressed by lam et al in [6].They developed an efficient algorithm that solves the problem in $O(n^2)$ time for n-nodes expression tree.

The problem addressed in this paper is a generalization of the work presented in [6] in which the processing time required by the data objects are not necessarily the same. Moreover, our formulation is extended to cover the case of evaluation data objects on multiple processors with shared memory. In addition, this paper considers the problem of memory-usage optimization in case the given computation is represented by a directed acyclic graph (DAG) not only the simpler tree representation case. That turns the problem to be NP-Complete [7].

We are proposing a mathematical formulation solution for the memory usage optimization problem. We present a mixed integer linear programming (MILP) formulation for the two problems stated above to obtain the exact solution.

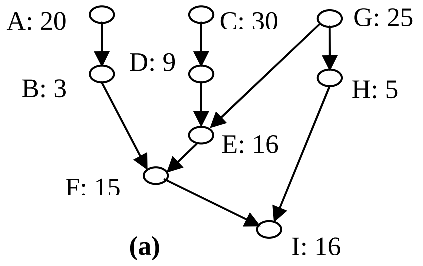
Notations

M(i) = memory size needed for data object *i*.
D(i) = execution time (in number of time units) needed for data object *i*.
ex-M(i) = extended memory size needed for data object *i* and all of its children.

λ = total number of time-steps.
mem_j = total memory usage by all active nodes at step j.
$R(i)$ = Time-frame of node i, which is the set of time-steps that start at its earliest time, *ASAP*, and end at its latest time, *ALAP*.
mem_constraint = maximum memory space allowed.

II. Preliminaries

In case the graph representation of the problem is a DAG, transitive reduction is applied. For example if there are edges *(a,b)*, *(b,c)* and *(a,c)*, then edges *(a,b)* and *(b,c)* do not contribute to the memory usage because they are dominated by the edge *(a,c)* which defines the life-time for node a (times at which data object a is still occupying a space in memory). The edge *(a,c)* is called the live-range edge in this case. After all the edges dominated by live-range edges are eliminated, a dummy node, v_d is introduced if there is some node v with more than one outgoing edge. Edges are classified according to their usage in the formulation and they are given a type from the set {0, 1, 2, 3}. All the edges are initially classified as type-0 edges. For each node v with more than one outgoing edge, we introduce a live-range edge between node v and its dummy node v_d. Such edges (v, v_d) are classified as type-3 edges. For each node w, a successor of node v, edge (v, w) is called a *reflexive edge* and it is marked type-1. Then for each reflexive edge (v, w), we introduce a dummy edge (w, v_d) and mark this edge as a type-2 edge. In Figure 1-(a), node G has two outgoing edges *(G,E)* and *(G,H)*. Thus, a dummy node G_d is added as well as two dummy edges (E,G_d) and (H,G_d) for the reflexive edges *(G,E)* and *(G,H)*, respectively, as shown in Figure 1-(b).



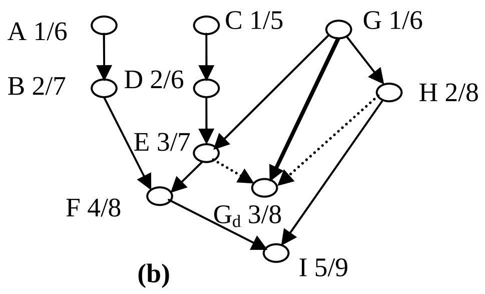


Figure 1: Sample DFG for memory evaluation: **(a)** original DFG, **(b)** the preprocessed DFG.

Based on the definitions of edge sets given above, edges of type-3 generate redundant inequalities if they are considered in the precedence constraints; and edges of type-1 and type-2 do not contribute in the memory usage because their role is taken by type-3 edges. Thus, type-3 edges are not to be considered in the precedence constraints while type-1 and type-2 edges are not to be considered in the memory constraints.

For mixed integer linear programming formulation, we introduce a 0-1 (unknown) variable, x_{ij}, that takes a value 1 if node i starts to be evaluated at time-step j, and 0 otherwise. Equation (1) below is developed to precisely define the memory usage at time-step j, mem_j. It considers the contribution of node i in memory usage during the set of time-steps starting directly after it finishes its processing until its last parent l is processed completely, which is captured by the first term inside the summation over the DFG edges E. In addition, the memory contribution of a node j to the memory usage while it is being processed is considered by the second term inside the summation over the set V of the DFG nodes:

$$mem_j = \sum_{(i,l)\in E} \left\{ M(i) \left(\sum_{k=1}^{j-D(i)} x_{ik} - \sum_{k=1}^{j-D(l)} x_{lk} \right) \right\} + \sum_{i\in V} \left(M(i) \sum_{k=j-D(i)+1}^{j} x_{ik} \right) \quad (1)$$

III. Evaluation Order and Performance-Constrained Evaluation (EOPCE)

Given a DFG representation of the evaluation order problem *G(V, E, M)*, the execution time needed for each node *D*, and the total time allowed to finish the computations (in number of time units) λ, the goal is to find an evaluation order for these data objects that achieves the least memory usage. The formulation is as follows.

- **Objective Function**

The objective is to minimize memory usage required at any time during the DFG evaluation process. This amount of memory space is expressed as the maximum memory usage at any time-step. Thus, the objective function is expressed as:

Minimize: *mem,* (2)

where *mem* will be used as a variable to be evaluated.

- **Uniqueness Constraints**

Each node should start at exactly one time-step within its time-frame. Inequality (3) is used to model this constraint.

$$\sum_j x_{ij} = 1 \qquad \forall i,\ j \in R(i) \qquad (3)$$

- **Precedence Constraints**

The fact that a data object can start to be evaluated only after all its data inputs (children) are ready is modeled as a precedence relation, one for each child-parent pair as shown in inequality (4), where each summation term in the left-hand side expresses the time-step at which a data object starts execution. Note that, in the case where the evaluation order is the objective, the delay, *D(i),* is treated as one.

$$\sum_{j\in R(i)} jx_{ij} - \sum_{j\in R(l)} jx_{lj} \leq -D(i) \quad \forall (i,l)\in E \qquad (4)$$

- **Memory Constraints**

The maximum memory space required during computation process is modeled by using a single variable that constrains total memory usage of all active nodes at each time-step as shown in inequality (5) and then minimizing that variable as the objective function.

$$\sum_{(i,l)\in E} \left\{ M(i)\left(\sum_{k=1}^{j-D(i)} x_{ik} - \sum_{k=1}^{j-D(l)} x_{lk} \right) \right\} + \sum_{i\in V} \left(M(i) \sum_{k=j-D(i)+1}^{j} x_{ik} \right) - mem \leq 0. \quad \forall j\in[1,\lambda] \qquad (5)$$

In case the objective is the evaluation order without any constraint on the time needed, the execution time for each node is treated as one. In addition, another constraint is needed to force that at each time-step exactly one node is evaluated. This can be modeled as Equation (6).

$$\sum_{i\in V} x_{ij} = 1 \qquad \forall j\in[1,\lambda] \qquad (6)$$

IV. Memory-Constrained Evaluation

Given a DFG representation of the evaluation order problem *G(V, E, M)*, the execution time needed for each node *D*, and the maximum memory space allowed, *mem_constraint*, the goal is to find an evaluation order for these data objects that minimizes the total execution time needed.

The mathematical formulation is similar to the EOPCE problem described above. Uniqueness and precedence constraints are as described in inequalities (3) and (4), while the memory constraint is different from the one in inequality (5) in which the maximum memory is explicitly posed as a constraint by the input *mem_constraint* value rather than by a variable as shown in inequality (7). Note that λ_{ub} used in inequality (7) is an upper-bound estimation of the total execution time. The total execution time, λ, is used twice as a variable, one to pose a constraint over the start time-step of the last node in the DFG (node without successors) as shown in inequality (8) and the other as the objective function to be minimized as shown in Equation (9).

$$\sum_{(i,l)\in E} \left\{ M(i)\left(\sum_{k=1}^{j-D(i)} x_{ik} - \sum_{k=1}^{j-D(l)} x_{lk} \right) \right\} + \sum_{i\in V} \left(M(i) \sum_{k=j-D(i)+1}^{j} x_{ik} \right) \leq mem\_\text{constraint}. \quad \forall j\in[1,\lambda_{ub}] \qquad (7)$$

- **Time Constraints**

$$\sum_{j} (j + D(i) - 1)x_{ij} \leq \lambda . \qquad \forall\ node\ \ i\ \ without\ \ succesors \qquad (8)$$

- **Objective function**

Minimize: *Total execution time*, λ (9)

V. Design Space Exploration

For design space exploration (DSE), we have adopted an efficient strategy in doing the experiments that can precisely identify the *pareto points* in the design space. A point in the design space that achieves the least memory usage for a given time constraint, while the time constraint is the minimum execution time for that memory usage is called a *pareto point*. This strategy is depicted in Figure 2.

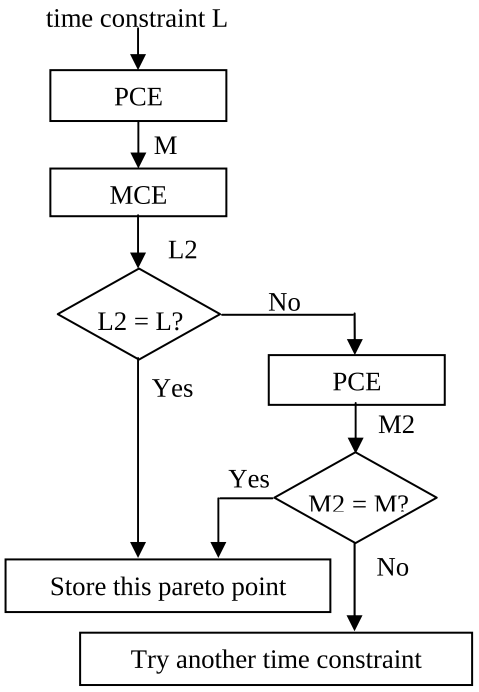


Figure 2: Flow chart for DSE strategy.

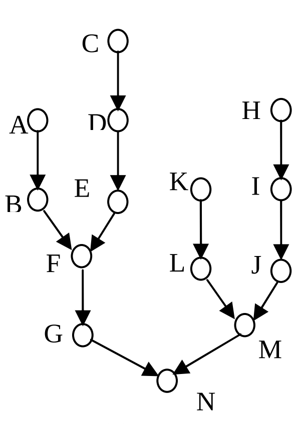
First, for a given time constraint L, we solve the PCE problem to find the least memory usage M. Then this memory M is used as a constraint in solving the MCE problem to find the minimum total execution time L2. If that execution time L2 is the same as the time constraint L, then this point (L, M) is a pareto point, otherwise L2 is fed back to the PCE as a constraint to find the least memory usage M2. Again, if this resultant memory M2 is the same as M, then (L2, M) is a pareto point.

In DSE, the execution time taken by the ILP solver for each experiment as well as the total execution time needed to fully explore the design space depends on many factors. It depends on the DFG structure, how far the time constraint is from the critical path length, the number of nodes in the DFG, as well as the distribution of memory costs of the DFG nodes. The first two factors define the number of variables associated with each node while the number of DFG nodes contributes to the total number of variables. The memory cost distribution of DFG nodes as well as the DFG structure shape the search space in the ILP solver for optimal objective function and hence the solution time.

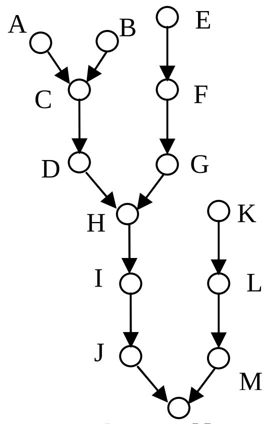
VI. Experimental Results

We have used five diverse examples to work as benchmarks in testing our presented ILP formulations for memory evaluation problem. These test examples are Exdag1 shown in Figure 1, Ex1 that is a tree structure of Exdag1 after the edge (G, E) is removed, Ex2 that is the same as Ex1 but with different memory cost for each node, and Ex3 and Ex4 presented in Figure 3-(a) and 3-(b), respectively. Ex3 and Ex4 have the same number of nodes, memory costs, and execution time, but are different in the graph structure. The Experiments take place on the *SUN ENTERPRISE 4500* workstation. This workstation has eight 333MHz SPARC CPU's and 2GB RAM, and it works with SOLARIS 8 operating system.

Results for memory evaluation under timing constraints when the computation elements (nodes) are assumed to have unit time delays are shown in Table 3, while the results for the general case in which the computation elements can take different times to be processed are shown in Table 4. The optimal ILP solution (maximum memory usage in case of PCE problem and the number of time units in case of the MCE problem) is tabulated under the heading "ILP". In the result tables, "L" is the time constraint, "mem" is the memory constraint, and "time" is the ILP solution time in seconds. Experiments are done for different time-constraints ranging from the critical path length to twice the critical path length in the general case and for a time-constraint equaling the number of nodes in the case of a unit time delay.



(a)



(b)

node	A	B	C	D	E	F	G
M	2	20	12	9	11	7	9
d	3	2	1	1	2	1	1
node	H	I	J	K	L	M	N
M	5	12	16	11	13	5	8
d	2	1	2	2	1	2	1

(c)

Figure 3: Sample examples for memory evaluation: **(a)** Ex3, **(b)** Ex4, **(c)** memory size, M, and delay, d, for each node in (a) and (b).

Table 3: Performance constrained evaluation with unit delay

(a) Ex1

L	ILP	time
9	39	0.44
8	45	0.20
7	53	0.11
6	59	0.05
5	64	0.02

(b) Ex2

L	ILP	time
9	44	0.38
8	47	0.21
7	54	0.17
6	59	0.05
5	75	0.01

(d) Exdag1

L	ILP	time
9	50	0.38
8	53	0.22
7	58	0.11
6	59	0.04
5	84	0.01

(d) Ex3

L	ILP	time
14	43	16.2
10	46	1.03
8	54	0.19
7	61	0.01
6	79	0.02

(d) Ex4

L	ILP	time
14	34	11.4
12	34	2.7
10	35	0.79
8	46	0.08
7	52	0.01

Optimal ILP results for memory-constrained evaluation are presented in Table 5 for different values of memory constraints for each test example. Results show that the ILP solution time is a fraction of second and it is large only when the time constraint is far from the length of the critical path and when the memory constraints are very stringent.

Table 4: Performance constrained evaluation

(a) Ex1

L	ILP	time
20	39	5.53
15	53	0.75
12	64	0.08
11	67	0.03
10	78	0.03

(b) Ex2

L	ILP	time
20	44	8.35
18	47	2.2
15	57	1.0
12	59	0.12
10	84	0.06

(c) Exdag1

L	ILP	time
20	50	28.5
15	58	0.81
12	67	0.08
11	78	0.08
10	78	0.03

(d) Ex3

L	ILP	time
16	43	7.18
14	47	1.58
12	50	0.46
10	67	0.09
8	83	0.02

(e) Ex4

L	ILP	time
22	34	28.5
16	35	4.0
14	41	0.62
12	46	0.13
11	46	0.03

Table 5: Memory constrained evaluation

(a) Ex1

mem	L	time
39	20	4.9
48	17	3.7
50	17	3.7
64	12	0.11
78	10	0.04

(b) Ex2

mem	L	time
44	20	5.0
47	17	2.3
57	15	1.56
60	12	0.13
84	10	0.03

(c) Exdag1

mem	L	time
50	20	9.5
53	17	2.8
67	12	0.26
70	12	0.12
78	10	0.04

(d) Ex3

mem	L	time
43	16	4.9
47	14	2.5
50	12	0.88
67	10	0.22
83	8	0.03

(e) Ex4

mem	L	time
34	17	6.8
35	16	1.94
41	13	0.37
46	11	0.02

VII. Conclusion

In this paper, we have addressed the problem of memory usage optimization in the evaluation of computations with large data objects. We are interested in the situations where the data objects are so large to fit in memory that they have to be dynamically allocated and deallocated during expression evaluation. We addressed the problem of finding an evaluation order for these data objects to achieve the least memory usage as well as its two variations in the case of multiple-processors using shared memory. These problems are the performance-constrained evaluation (PCE) and memory-constrained evaluation (MCE). We have developed an ILP formulation to optimally solve the three problems. Solution results show that the ILP solution time is small especially when the time constraint is close to the critical path length or when the memory constraint is less stringent.

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

[1] W. Aulbur, *Parallel Implementation of Quasi-Particle Calculations of Semiconductors and Insulators*, Ph.D. Dissertation, Ohio State University, Oct. 1996.

[2] M. S. Hybertsen and S. G. Louie, “Electronic Correlation in Semiconductors and Insulators: Band Gaps and Quasi-particle Energies,” *Phys. Rev. B*, 34, pp. 5390, 1986.

[3] 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, pp. 1827, 1995.

[6] C. Lam, D. Cociorva, G. Baumgartner, and P. Sadayappan, “Memory-Optimal Evaluation of Expression Trees Involving Large Objects,” Technical Report OSUCISRC-5/99-TR13, Dept. of Computer and Information Science, The Ohio State University, May 1999.

[4] I. Nakata, *On compiling algorithms for arithmetic expressions*, Comm. ACM, Vol.10, pp. 492–494, 1967.

[5] R. Sethi, J. D. Ullman, *The generation of optimal code for arithmetic expressions*, J. ACM, 17(1), pp. 715–728, October 1970.

[7] R. Sethi, *Complete register allocation problems*, SIAMJ. Computing, 4(3), pp. 226–248, September 1975.