A FUZZY APPROACH TO AUTOMATIC DATA LOCALITY OPTIMIZATION

Claudia Leopold
Fakultät für Mathematik und Informatik
Friedrich-Schiller-Universität Jena
07740 Jena, Germany
claudia@minet.uni-jena.de

Keywords: fuzzy relations, fuzzy clustering, locality, memory hierarchies, compilation

ABSTRACT

Many programs, both in sequential and parallel computation, can be significantly speeded up by increasing their degree of locality, i.e., by storing data that are used together in the same block, and by ordering the statements to maximize reuse of local data. Based on the framework of a two-level memory in sequential computation, this paper suggests an approach to automatic locality optimization that operates at the level of statement instances, and can handle irregularity. The approach combines conservative data dependency constraints with fuzzy knowledge on the relatedness of individual data and operations. The optimization is an iterative process, where data distribution and statement ordering are alternatingly refined, and intermediate results are fuzzy.

1 INTRODUCTION

Due to the increasing gap between processing speeds on one hand and memory and network speeds on the other, the running time of large-scale programs is increasingly dominated by data transmission times. This is a problem of both sequential and parallel computation, and occurs at several levels of the memory hierarchy, in particular

"Permission to make digital/hard copy of all or part of this material without fee is granted provided that copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery, Inc.(ACM). To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee."
© 1996 ACM 0-89791-820-7 96 0002 3.50

- between cache and main memory (in sequential and parallel computers)
- between main memory and secondary memory like disks (in sequential and parallel computers)
- between local and global memory in shared memory parallel computers, and
- between local and remote memory in distributed memory parallel computers.

In this paper, we restrict considerations to a simple two-level memory model in the context of sequential computation. It is expected, however, that similar techniques should be helpful to reduce the number of data transfers in the above listed more involved cases, too.

The two-level model divides memory into a fast module with limited capacity, and a slow module with potentially infinite capacity. Data are stored in fixed-sized blocks. Whenever some data element is requested by the CPU, its block is moved to the fast memory (if it is not already there). If the fast memory is full, the *L*east *R*ecently *U*sed block is replaced (LRU scheme). The aim of program optimizations in the two-level model is to *minimize the number of block transfers.*

Optimizations typically work by increasing the degree of ***temporal and spatial locality*** of the programs. Temporal locality means that accesses to the same data element should be clustered in time, spatial locality means that successively accessed data should be stored in the same block. Locality is a property of accesses to both data and code, only data locality is considered here.

Automatic locality optimization is typically done in a compiler, that has several possibilities:

- data transformations change the distribution of data to blocks
- code transformations change the order of statements, and
- modifications of the memory management use a more knowledgeable replacement scheme than LRU (and can also improve the overlap between communication and computation by prefetching)

In this paper, we suggest an approach that combines data and code transformations; modifications of the memory management are not yet considered for simplicity. Briefly, the approach operates on the level of individual statement instances and data elements. Via profiling, we find out how often particular pairs of data are used together. The corresponding fuzzy relation is represented by a graph that is partitioned using fuzzy clustering methods. The result is a preliminary distribution of data to blocks, represented by fuzzy membership values. It is used to derive an initial ordering of the statements which in turn is used for refining the data distribution. The process is iterated.

The current paper is a report about ongoing work. Many details are still left open, or are given preliminary answers only, that have to be complemented and refined in course of further research.

The paper starts with an overview on related work in section 2. Then our approach is outlined in sections 3 and 4. While section 3 discusses general questions of locality optimizations at the level of statement instances, section 4 describes the use of fuzzy methods. Section 5 finishes with conclusions.

2 RELATED WORK

Code transformations that among other purposes are useful for locality optimization have received a lot of attention in the literature (for an overview see [1]). A characteristic feature of this approach is the consideration of nested loop structures, with few statements, that operate on large arrays, and access the arrays via simple index expressions. Program optimization is based on applying a sequence of loop transformations, chosen from a large but limited set, to the performance-critical loops of the program. The approach is successful in practice, but only for loop structures with a high degree of regularity.

Data transformations have mainly been studied in the context of data-parallel computing, for distributing arrays among processors (see e.g. [6]). They have also been investigated in the field of data bases, to minimize I/O ([7]).

Locality can further be improved via pattern matcl ing, where program segments are substituted by semai tically equivalent counterparts.

In the past, data and code transformations have bee typically investigated in separation, it was advocate only recently that they should be combined ([4]).

Automatic locality optimization is recognized as difficult field, hence many practical systems rely on hel from the programmer. In languages like High Perfo mance Fortran, e.g., the programmer is engaged to spe ify a data distribution.

Despite of practical successes, automatic locality o timization can not be expected to find programs th use the minimum number of block transfers required f a problem, at least not in a foreseeable future. The re son is that the transformations accessible by compile — data redistributions and statement reorderings — a alone not sufficient to achieve optimal locality. In [examples are given where clever ideas and mathema ical insight are necessary to find optimized program that consist not only of reordered but of different stat ments, operating possibly on additional data structure Although consideration of data locality issues in alg rithm design and programming ([3]) is important, it also a burden for programmers. Hence, we think th automatic locality optimization is worth further inve tigation.

3 OUR APPROACH

We operate on an input program that consists statement instances only, i.e., loops have been con pletely unrolled and procedures have been inline While programs with statically determined control flo can be easily brought into this form, many progran have variable loop bounds, or the input size is not know in advance. Here, a conservative transformation into t desired form may be possible, if loop bounds and inp size can be bounded from above.

But even if the input size is not known (or comput tional complexity does not allow working with the re size), our approach might be of help. In these cases, v suggest optimizing the program for several reasonab small exemplary input sizes, hopefully common cha acteristic features can be extracted from the solutio and generalized. The generalization will surely put lo of difficult questions, that are not yet tackled with t present paper. The idea is inspired by algorithm desig where it is often useful to start with designing solutio for small problem instances.

Often, only some few program segments are critical to performance, and it may suffice to consider them in separation.

Our optimization algorithm derives data distributions that may be not regular at all, hence requiring an involved address translation, that is acceptable only if the data elements are large. Otherwise, the derived distributions should be approximated by more regular ones that make address translation easier. The strive for regularity can be taken into account in the optimization algorithm, by increasing the relatedness values for structurally neighboured data.

We consider two kinds of information: data dependency constraints, and relatedness preferences. The former define a dependency graph that restricts the order of statement execution. It is a directed acyclic graph (acyclic, as we have unrolled loop structures), and will be denoted *dag*. Techniques for deriving the dag from the program are well-known ([1]), and we will just rely on them. They assume a dependence whenever it is possible, even if it is unlikely. We strictly follow the dag to *guarantee* correctness. Locality, in contrast, is an efficiency issue, hence approximate solutions, based on the processing of fuzzy relatedness preferences, are appropriate.

4 OPTIMIZATION ALGORITHM

Step 1: Transform the input program into the desired form and fix the input size. Determine the dag.

Step 2: Execute the program for several exemplary input suites, thereby record:

(a) the number of times each statement is carried out, and

(b) for each pair of a statement and a data element, the number of times the statement accesses the data element

Afterwards, divide the obtained values by the number of profile runs carried out, to scale them into range [0,1]. The scaled values are refered to as execution frequencies (in case (a)), and as relation $SD2$ (in case (b)). (The notation $SD2$ reflects that we have a fuzzy relation between statements (S) and data (D) established in step 2.)
Comment: The input suites must be provided by the user, and should be as representative as possible. If it is not possible to choose reasonable suites for the given input size, our optimization algorithm can not be applied.

Step 3: Derive a fuzzy relation $DD3$ from $SD2$ that reflects the preference of data pairs for being stored in the same block. It is derived via $DD3 := SD2^{-1} \circ SD2$, where $\circ$ is a suitably chosen fuzzy composition operator. $DD3$ defines an undirected weighted graph, with data elements as nodes and edges marked with the membership values of the relation. Distributing data to blocks now appears as a graph partitioning problem. Graph partitioning is NP-complete, but lots of approximation algorithms have been suggested (for an overview see [5]). In our context, the fuzzy c-means clustering algorithm FCM ([8], [9] pp. 227ff), may be in a modified form, seems to be a good choice, since it determines a distribution that is represented by fuzzy membership values. As the data distribution is an intermediate result, fuzziness permits transferring more knowledge into the next step than what would be possible otherwise. The FCM algorithm divides the graph into c clusters (blocks), by alternatingly refining the choice of cluster centers and the assignment of data to clusters. We can choose c to be the minimal number of blocks required from the cardinality of the data set, but we can also try larger values and pad the blocks, which sometimes pays off ([7]). The FCM algorithm was originally invented in an other context, and does not aim at constructing (approximately) equal-sized blocks. To include this issue of our application, adapt the membership values, e.g. $\mu_{ik}^{DB3} := (\mu_{ik}/card_i)/(\sum_j(\mu_{jk}/card_j))$, where μ_{ik}^{DB3} is the adapted membership value of the k-th data element to the i-th block (defining a fuzzy relation, $DB3$), μ_{ik} is the membership value of the k-th data element to the i-th block determined by FCM, and $card_i = \sum_l \mu_{il}$ is the cardinality of the i-th block.

Step 4: Order the statements using the data distribution from step 3.

Step 4.1: Compose $SD2$ and $DB3$ according to $SD2 \circ (DB3 \circ DB3^{-1}) \circ SD2^{-1}$ (where $\circ$ is a suitably chosen fuzzy composition operator) into a fuzzy relation on pairs of statements, SS4, that reflects their degree of preference for being executed close to each other. The subcomposition $DB3 \circ DB3^{-1}$ should yield a relation that takes value 1 for pairs of identical data.

Step 4.2: Determine a maximal length path through the dag, with length defined as the sum of the execution frequencies. Initialize the statement sequence representing the program with the statements of this path.

Step 4.3: Successively add all other statements to the sequence.
(a) Choose the next statement to be placed heuristically, e.g. as the one with the strongest dag connection to the already placed statements. Here, strength can be determined as the number of (forward and backward) edges in the dag between the new statement and the already

placed statements, taking into account their execution frequencies.
(b) If the dag dependencies let a choice, decide between the possible placements for the new statement on the basis of relation $SS4$. Note that a placement may imply both an increase in locality (if the new statement reuses data of the surrounding statements) and a decrease in locality (if the new statement separates surrounding statements such that the fast memory's capacity does not permit reuse between them anymore). The decision for a placement should take into account both aspects.

Step 5: Now that we have an approximate ordering of the statements, the data distribution can be refined to store preferably in the same block not only data that are accessed in the same but also those that are accessed in successive statements.
(a) Construct a relation between statements, $SS5$, reflecting their closeness in the statement sequence. The membership values to the relation should be the higher the closer the corresponding statements are placed, the higher their execution frequencies are, and the larger the capacity of the fast memory is.
(b) Replace relation $DD3$ by relation $DD5$, determined according to $DD5 := SD2^{-1} \circ SS5 \circ SD2$, and apply the FCM algorithm to the graph of $DD5$.

Step 6: Iterate over steps 4 and 5 until either the gain in performance between iterations is small, or the computing time one is willing to spend is exhausted. For the former criterion, a performance prediction method is needed. We suggest to simulate the program, thereby determining the chance of an access to require a block transfer on the basis of the fuzzy values.

Step 7: Defuzzify the data distribution, and finish with a last refinement of the statement ordering.

5 CONCLUDING REMARKS

This paper has introduced the idea of an automatic locality optimization method that uses fuzzy relations between statement instances and data elements. Work along this approach has only begun, and there are lots of issues that have to be addressed by further research. In particular, many important details have yet to be supplemented, e.g. how to compose the relations and how to defuzzify the data distribution. Refinements, modifications and extensions will be necessary to improve both the quality of the solutions and the time requirements of the algorithm. Experiments are needed to find out if the approach is successful in practice. Comparing the approach to other methods, it seems to be more general and flexible, but on the backside has a higher computational complexity. It is not intended as a substitute for other techniques but rather as a complemei for cases where other techniques are not applicable c do not produce sufficient results: indirect accesses, con plicated index expressions and irregularity that can nc be covered by predefined transformation patterns.

References

[1] D.F. Bacon, S.L. Graham, O.J. Sharp,"Compil Transformations for High-Performance Compu ing", *ACM Computing Surveys*, 26(4), pp. 345-42 1994

[2] S. Carr,"Memory-Hierarchy Management", Ph.I thesis, Rice University, 1993

[3] Y.-J. Chiang, M.T. Goodrich, E.F. Grove, I Tamassia, D.E. Vengroff, J.S. Vitter,"Externa Memory Graph Algorithms", *Proc. ACM-SIA Symposium on Discrete Algorithms*, pp. 139-14 1995

[4] M. Cierniak, W. Li,"Unifying Data and Contr Transformations for Distributed Shared Memo Machines", *Proc. ACM SIGPLAN'95 Conferen on Programming Language Design and Impleme tation*, 1995

[5] G. Karypis, V. Kumar,"A Fast and High Qua ity Multilevel Scheme for Partitioning Irregul Graphs", TR 95-035, University of Minnesot 1995

[6] U. Kremer, K. Kennedy "Automatic Data Layo For High Performance Fortran", *Workshop on A tomatic Data Layout and Performance Predictio* 1995 http://www.cs.rice.edu/ kremer/AP95

[7] M.M. Tsangaris, J.F. Naughton "A Stochastic A proach for Clustering in Object Bases", *Proc. 19 ACM SIGMOD Conference on the Management Data*, pp. 12-21, 1991

[8] J.-T. Yan, P.-Y. Hsiao,"A Fuzzy Clustering Alg rithm for Graph Bisection", *Information Proces ing Letters*, 52, pp. 259-263, 1994

[9] H.-J. Zimmermann, "Fuzzy Set Theory and its A plications", 2nd edition, Kluwer Academic Publis ers, 1991