An Automatic Hierarchical Data Placement Method [*]

Chengliang Zhang, Yutao Zhong, Chen Ding and Mitsu Ogihara

Computer Science Department
University of Rochester
{zhangchl,ytzhong,cding,ogihara}@cs.rochester.edu

1 The problem

It is well known that the memory of most machines is organized as a hierarchy. To fully utilize this memory hierarchy, *hierarchical data placement* reorganizes program data into many layers of data blocks to exploit data locality at *all* memory levels.

The data placement problem can be deemed as a mapping problem. The domain is the set of programs, which is the power set of the set of all possible sequences of data accesses. The image is all data decompositions. The mapping takes a program to its hierarchical data placement; a uniform address space to a data hierarchy; and sequences in time to structures in space.

In general, the optimal data placement problem is an NP-hard problem [1]. However, there are specialized data placement methods designed for important applications such as matrix operations, N-body simulation and search trees. A question arises naturally: can we find the data placement methods automatically?

We propose to use *reference affinity* to solve this problem. Reference affinity measures whether a group of data is always accessed together during an execution. It gives a unique and hierarchical partition of the data [3].

2 Our solution

Starting from a trace, which is a sequence of accesses to a set of data elements, let's suppose we have reference affinity groups at hand. Depending on how strong the affinities are, we can divide the reference affinity groups into two cases: groups having constant distances and those having variable distances.

For the constant distance case, the construction of the data hierarchy involves a bottom-up traversal of the affinity hierarchy.

For the variable distance case, simply putting them into one big group is a bad idea since we can not guarantee the full group utilization as we do for constant case. Instead, we bottom-up partition the probable affinity graph, where probable affinity measures how frequent a pair of data elements is accessed together.

3 Justification

From our analysis, the method described above can automatically give the desirable data hierarchy not only for the diverse cases reported by past studies—Morton layout for matrices, Hilbert curve for particles, and van Emde Boas layout for search trees—but also for important new cases—random accesses and random walks [2]. Hopefully it can establish the subtle link between the reference affinity model and the data placement problem. These results advance our understanding of the hierarchical data placement and open the door for research into more effective and efficient placement methods for general-purpose programs. The strong relation between the access pattern in computation and the spatial relation in data gives us a basis for improving the programming, compiler, and language support for modern computer memory systems.

References

[1] E. Petrank and D. Rawitz. The hardness of cache conscious data placement. In *Proceedings of ACM Symposium on Principles of Programming Languages*, Portland, Oregon, January 2002.

[2] C. Zhang, Y. Zhong, C. Ding, and M. Ogihara. A method for hierarchical data placement. Technical Report TR 845, Department of Computer Science, University of Rochester, Aug 2004.

[3] Y. Zhong, M. Orlovich, X. Shen, and C. Ding. Array regrouping and structure splitting using whole-program reference affinity. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2004.

[*] This work is under review for publication by POPL