Sparse Tiling for Unstructured Mesh Applications in the OP2 Framework

Fabio Luporini
Department of Computing
Imperial College London
London, UK
Email: f.luporini12@imperial.ac.uk

Paul H. J. Kelly
Department of Computing
Imperial College London
London, UK
Email: p.kelly@imperial.ac.uk

Carlo Bertolli
IBM TJ Watson Research
New York, USA
Email: cbertol@us.ibm.com

***Abstract*—Unstructured meshes are widely-used in scientific computing for implementing numerical methods. In applications based on such abstraction parallelism is well-exposed, but potential data reuse between different loops is, in general, lost. This is mainly due to the presence of indirect memory accesses (like `A[B[i]]`) that prevent many optimizations.**

Sparse tiling is a well-known code transformation that aims at obtaining data locality across different loops by scheduling sets of iterations determined at run-time that share some blocks of data. However, so far it has been "manually" applied to small benchmarks only. In this work we show our on-going research into integrating sparse tiling with the OP2 framework for unstructured mesh computations. Early experiments with the Airfoil benchmark show that sparse tiling improves the run-time performance of 20% for both the sequential and OpenMP versions. Evaluating sparse tiling in industrial applications supported by OP2 is the ultimate goal.

I. Sparse Tiling in the OP2 Framework for Unstructured Mesh Computations

Unstructured mesh are widely-used in scientific computing for implementing numerical methods like Finite Elements or Finite Volume. Applications based on unstructured meshes commonly require long execution times, with intensive data access. They usually feature sequences of loops that iterate on sets of mesh entities (e.g. edges, cells) and share data through another set of mesh entities (typically vertices). There are no constraints on the order with which iterations of a loop need to be executed, so parallelism is always well exposed (and exploited in real applications), but data locality across subsequent loops is, in general, lost. This is mainly due to the presence of indirect memory accesses (like `A[B[i]]`) that prevent many optimizations. A solution to this problem is to use communication avoiding code transformations, like sparse tiling [2], which aim at obtaining data locality across different loops by scheduling sets of iterations determined at run-time that share some blocks of data. However, so far they have been "manually" applied to small benchmarks only. In this paper we present a general algorithm to automate sparse tiling in a framework for unstructured mesh computations and evaluate it in real-world applications.

The OP2 framework [1], which has been successfully used for developing performance-critical, industrial applications, targets computations based on unstructured meshes. In OP2 an unstructured mesh can be constructed in terms of sets, relationships between sets and datasets. A programming construct called *op_par_loop* is used to modify datasets by applying a function to all elements of a set. As an example, a loop over a set of edges can modify the weight of an edge by reading (through indirect memory accesses) values from the adjacent vertices, for each element in the set. According to the execution model of an OP2 parallel loop, iterations can be carried out in any order and/or completely in parallel. Data dependencies between iterations of the same loop are handled by the OP2 run-time support, through techniques that prevent data races. OP2 applications, therefore, are composed of "chains" of parallel loops that work on mesh entities.

The difficulty of sparse tiling in unstructured mesh applications comes from the presence of indirect memory accesses. Tiling a chain of *op_par_loop*s means that instead of executing successive loops in a serial way (i.e. one after the other), tiles are suitably created including iterations from different loops. This implies that the implementation of a chain of loops schedules tiles, rather than iterations of a loop for each loop in the chain. Compared to a simpler execution model, in which an implicit global synchronization is present between successive parallel loops, this approach enables data reuse inside a tile.

The sparse tiling algorithm belongs to the class of Inspector/Executor strategies. At run-time, the "inspector" code starts by partitioning a set over which the loops in the chain share data. A partial ordering of partitions can be defined by colouring them (or "assigning a priority") in such a way that adjacent partitions are assigned different colours. At this point, each tile (or partition) is expanded by adding elements from the iteration set of each loop in the chain. The correctness of the computation is preserved by ensuring that a tile executes a particular element (e.g. vertex, edge, cell) only if all of its adjacent base set elements are already updated. This is achieved by taking advantage of the colour ordering. It is worth noticing that the sparse tiling algorithm is simplified by the inherent OP2 programming model. Nevertheless, it can be suitably modified to be used in more general contexts.

Early experiments with the OP2 Airfoil application, which were executed on different multi-core CPUs, show that sparse tiling improves the run-time performance of 20% in both the sequential and OpenMP versions. Evaluating sparse tiling in industrial applications supported by OP2 is the ultimate goal.

References

[1] M.B. Giles, G.R. Mudalige, Z. Sharif, G. Markall, and P.H.J. Kelly. Performance analysis and optimization of the op2 framework on many-core architectures. *Comput. J.*, 55(2):168–180, February 2012.

[2] Michelle Mills Strout, Larry Carter, Jeanne Ferrante, and Barbara Kreaseck. Sparse tiling for stationary iterative methods. *INTERNATIONAL JOURNAL OF HIGH PERFORMANCE COMPUTING APPLICATIONS*, 18(1):2004, 2004.