Cache-Blocking Tiling of Large Stencil Codes at Runtime

Istvan Z. Reguly
Pazmany Peter Catholic University
Budapest, Hungary
reguly.istvan@itk.ppke.hu

Gihan R. Mudalige
University of Warwick
Coventry, UK
g.mudalige@warwick.ac.uk

Michael B. Giles
University of Oxford
Oxford, UK
mike.giles@maths.ox.ac.uk

ABSTRACT

Stencil codes on structured meshes are well-known to be bound by memory bandwidth. Previous research has shown that compiler techniques that reorder loop schedules to improve temporal locality across loop nests, such as tiling, work particularly well. However in large codes the scope of such analysis is limited by the large number of code paths, compilation units, and run-time parameters. We present how, through run-time analysis of data dependencies across stencil loops enables the OPS domain specific language to tile across a large number of different loops. This lets us tackle much larger applications than previously studied: we demonstrate 1.7-3.5x performance improvement on CloverLeaf 2D, CloverLeaf 3D, TeaLeaf and OpenSBLI, tiling across up to 650 subsequent loopnests accessing up to 30 different state variables per gridpoint with up to 46 different stencils. We also demonstrate excellent strong and weak scalability of our approach on up to 4608 Broadwell cores.

KEYWORDS

Tiling, OPS, DSL, Performance

ACM Reference format:
Istvan Z. Reguly, Gihan R. Mudalige, and Michael B. Giles. 2017. Cache-Blocking Tiling of Large Stencil Codes at Runtime. In *Proceedings of Supercomputing 17, Denver, Colorado USA, November 2017 (SC'17),* 2 pages.
https://doi.org/

1 INTRODUCTION

Transformations to the scheduling of both individual loopnests and multiple loopnests have been studied for a long time; loop blocking [11] improves temporal locality in a single loopnest, fusion [4] merges the bodies of subsequent loops that access the same data, also improving temporal locality. Indeed, most high-performance compilers automatically apply similar, and more complicated transformations already. A comprehensive framework for developing, verifying and applying such transformations is the polyhedral framework [1, 5, 10]; significant amounts of research have been carried out demonstrating transformations to affine and some non-affine loop structures. One of the most well studied transformations is tiling, where data locality is improved across multiple loopnests; several compilers carry out such optimisations, such as Pluto [2, 3], R-STREAM [8], Pochoir [9]. A compiler approach has many advantages, particularly when a handful of loops repeat a large number of times (e.g. stencil loops within a time iteration loop). However, in large applications code is spread across many compilation units,and there are many execution paths not known at compile time, therefore in too many cases (such as in the applications we study) a compiler cannot exactly determine a long enough sequence of loops that would be worth tiling across.

This is the challenge addressed by our research: by using delayed evaluation of nested loops expressed in the OPS domain specific language, we can determine the exact sequence of the loops, and given access-execute information, we can also determine the data dependencies across loops, which lets us carry out tiling fully automatically.

SC'17, November 2017, Denver, Colorado USA

ACM ISBN ...$15.00
https://doi.org/

2 TILING IN OPS

OPS [6, 7] is a domain specific language for expressing computations on structured meshes. Using its abstraction, one can write applications by first defining structured blocks, a number of datasets defined on these blocks, and stencils used to access them. A parallel loop is defined by indicating the iteration range, specifying a computational kernel to be applied at each grid point, and the data accessed, including the stencil used and whether it is read, written, or both. Since all data is handed to the library, and data is only returned to the user through API calls, it is possible to use delayed evaluation and queue up computational loops until some data needs to be returned to the user (e.g. after a reduction). Given the sequence of loops, the datasets accessed, and the patterns of access, we can carry out similar transformations in terms of loop scheduling as the aforementioned compilers: in our work we apply a simple skewed tiling, and parallelise within the tiles with OpenMP. The size of the tiles is determined automatically by OPS given the number of datasets accessed and the last level cache size.

3 RESULTS

There is a number of larger stencil codes implemented with OPS, such as the CloverLeaf 2D/3D and TeaLeaf hydrodynamics codes from the Mantevo suite, and the OpenSBLI Navier-Stokes solver. These applications consist of thousands of lines of code, spread across many compilation units, and attempts to apply Pluto were either unsuccessful or did not result in meaningful performance improvements.

Utilising tiling in OPS does not require any changes to the user code, it simply has to be enabled at runtime. For CloverLeaf 2D and 3D, we tile over 150 and 600 loops respectively, and we get a 2× overall speedup on a single socket of a Xeon E5-2650 v3, achieving 66 and 52 GB/s throughput respectively. In TeaLeaf, where 2 loops repeat a large number of times, we see a 3.5× improvement and 164 GB/s, and in OpenSBLI with a particularly computationally expensive kernel, we gain a 1.7× speedup.

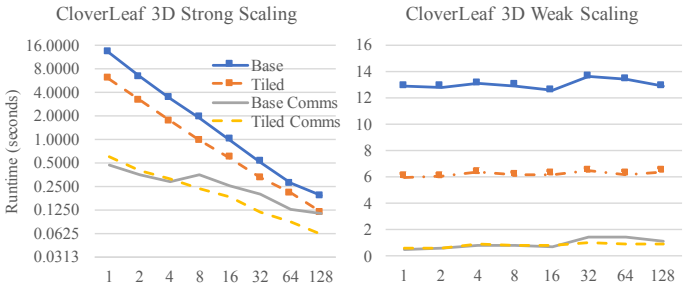

Figure 1: Scaling CloverLeaf 3D to multiple nodes on Marconi

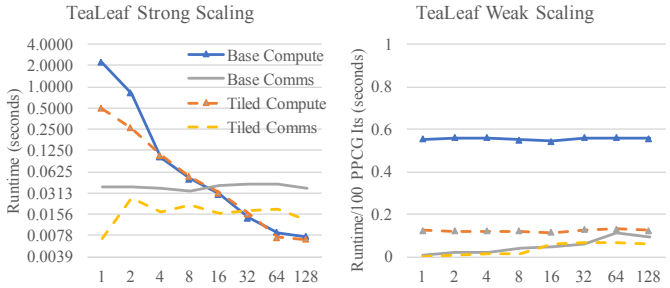

Figure 2: Scaling TeaLeaf to multiple nodes on Marconi

Strong and weak scaling on up to 128 nodes of the Marconi supercomputer (E5-2697 v4 CPUs) for CloverLeaf 3D and TeaLeaf are shown in figures 1 2 - excellent strong scaling can be observed up to the point where the problem size per CPU falls below the cache size, and the speedup is maintained when weak scaling.

REFERENCES

[1] Corinne Ancourt and François Irigoin. 1991. Scanning Polyhedra with DO Loops. In *Proceedings of the Third ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP '91)*. ACM, New York, NY, USA, 39–50. https://doi.org/10.1145/109625.109631

[2] Muthu Manikandan Baskaran, Nagavijayalakshmi Vydyanathan, Uday Kumar Reddy Bondhugula, J. Ramanujam, Atanas Rountev, and P. Sadayappan. 2009. Compiler-assisted dynamic scheduling for effective parallelization of loop nests on multicore processors. *SIGPLAN Notices* 44, 4 (February 2009), 219–228. https://doi.org/10.1145/1594835.1504209

[3] Uday Bondhugula, Muthu Baskaran, Sriram Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan. 2008. Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model. In *International Conference on Compiler Construction (ETAPS CC)*.

[4] Alain Darte. 1999. On the Complexity of Loop Fusion. In *Proceedings of the 1999 International Conference on Parallel Architectures and Compilation Techniques (PACT '99)*. IEEE Computer Society, Washington, DC, USA, 149–.

[5] Max Geigl. 1997. Parallelization of loop nests with general bounds in the polyhedron model. *Master's thesis, Universit at Passau* (1997).

[6] G. R. Mudalige, I. Z. Reguly, M. B. Giles, A. C. Mallinson, W. P. Gaudin, and J. A. Herdman. 2015. *Performance Analysis of a High-Level Abstractions-Based Hydrocode on Future Computing Systems*. Springer International Publishing, Cham, 85–104. https://doi.org/10.1007/978-3-319-17248-4_5

[7] István Z. Reguly, Gihan R. Mudalige, Michael B. Giles, Dan Curran, and Simon McIntosh-Smith. 2014. The OPS Domain Specific Abstraction for Multi-block Structured Grid Computations. In *Proceedings of the Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC '14)*. IEEE Press, Piscataway, NJ, USA, 58–67. https://doi.org/10.1109/WOLFHPC.2014.7

[8] Eric Schweitz, Richard Lethin, Allen Leung, and Benoit Meister. 2006. R-stream: A parametric high level compiler. *Proceedings of HPEC* (2006). http://llwww.ll.mit.edu/HPEC/agendas/proc06/Day2/21_Schweitz_Pres.pdf

[9] Yuan Tang, Rezaul Alam Chowdhury, Bradley C. Kuszmaul, Chi-Keung Luk, and Charles E. Leiserson. 2011. The Pochoir Stencil Compiler. In *Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '11)*. ACM, New York, NY, USA, 117–128. https://doi.org/10.1145/1989493.1989508

[10] Doran K. Wilde. 1993. *A Library for Doing Polyhedral Operations*. Technical Report 785. IRISA.

[11] M. Wolfe. 1989. More Iteration Space Tiling. In *Proceedings of the 1989 ACM/IEEE Conference on Supercomputing (Supercomputing '89)*. ACM, New York, NY, USA, 655–664. https://doi.org/10.1145/76263.76337