A general purpose automatic overlapped tiling technique in polyhedral frameworks

Jie Zhao

INRIA & École Normale Supérieure, 45 rue d'Ulm, 75005 Paris, France

jie.zhao@inria.fr

Abstract

Loop tiling [3] is an essential transformation to exploit data locality and parallelism, being implemented as a general technique in numerous existing polyhedral frameworks including Pluto [2], PPCG [7], etc. and domain specific automatic code generators like PolyMage [5]. Typically, automatic polyhedral frameworks either choose to implement the parallelogram tiling, avoiding the difficulty in performing code generation but leading to pipelined startup, or resort to sophisticated tiling shapes, e.g., diamond tiling [1], enabling concurrent startup but complicating both scheduling and code generation. Overlapped tiling [4] is a technique designed to eliminate pipelined startup by modifying tile shapes obtained from existing frameworks, but no implementations for general purpose multicores have been reported in a polyhedral framework except those domain specific code generators [5], preventing its application in general purpose languages and missing a comparison with other state-of-the-art techniques. We design and implement an automatic overlapped tiling technique for both general purpose and heterogeneous multicores in a polyhedral framework, neither being restricted to domain specific languages nor introducing sophisticated rescheduling in polyhedral frameworks. We first let a general polyhedral framework perform a parallelogram tiling and then expand the bounding faces of a tile by taking into consideration the constraints caused by inter-tile dependences, followed by expressing with well-defined nodes in a schedule-tree-based intermediate representation in general polyhedral frameworks, and finally achieve automatic code generation for overlapped tiling. We conduct some preliminary experiments on both stencil computations and image processing pipelines extracted from the PolyMage benchmarks but written in general purpose languages, validating our technique is comparable with the state-of-the-art diamond tiling on stencil computations by finely selecting tiling sizes without modifying the scheduler of a polyhedral framework and achieves the performance the same as or even better than PolyMage for image process pipelines without resorting to domain specific languages.

Introduction

- Being easy to implement, simple tiling shapes, e.g., parallelogram tiling, miss parallel startup.
- Enabling concurrent startup, complex tiling shapes complicate scheduling and code generation.

```
for (t=0; t<T; t++)
  for (i=1; i<N-1; i++)
    A[t+1][i]=0.25*(A[t][i+1]+2.0*A[t][i]+A[t][i-1]);
```

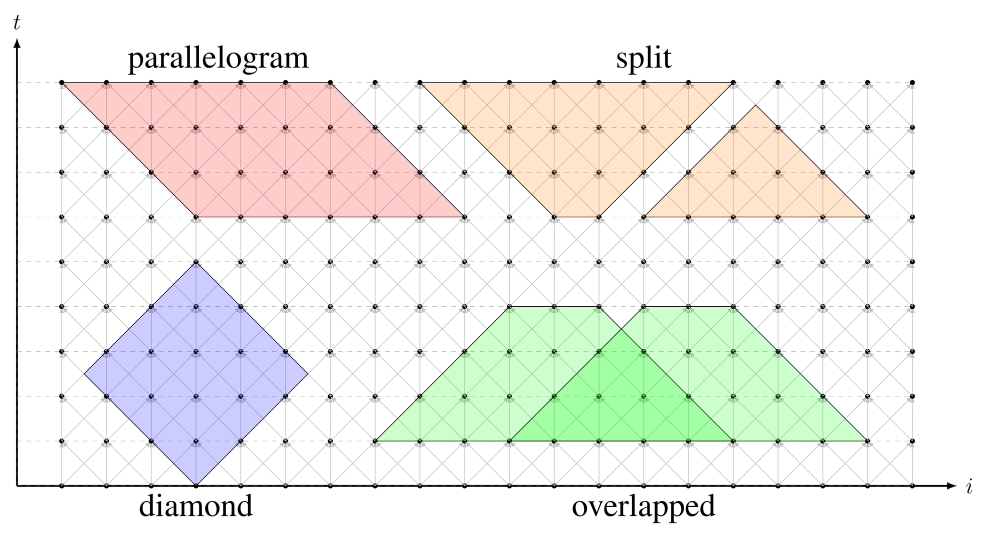

Figure 1: Tiling shapes

Table 1: Comparisons of different tiling techniques

	parallelogram	diamond	split	overlapped
shape complexity	×	✓	✓	✓
parallelism	×	✓	✓	✓
locality	✓	✓	✓	✓
redundancy	×	×	×	✓
algorithm complexity	×	✓	✓	×

- Our purpose—implement an effective tiling technique for programs written in general purpose languages:
  - Concurrent startup—~~parallelogram tiling~~
  - Different target architectures (CPUs and GPUs)—~~diamond tiling~~
  - Redundancy *v.s.* algorithm complexity—~~split tiling~~

Overlapped Tiling

```
for (i=1; i<N; i++)
    A[i]=f(i);
for(i=1; i<N-1; i++)
    B[i]=0.25*(A[i-1]+2*A[i]+A[i+1]);
for(i=2; i<N-2; i++)
    C[i]=0.25*(B[i-1]+2*B[i]+B[i+1]);
```

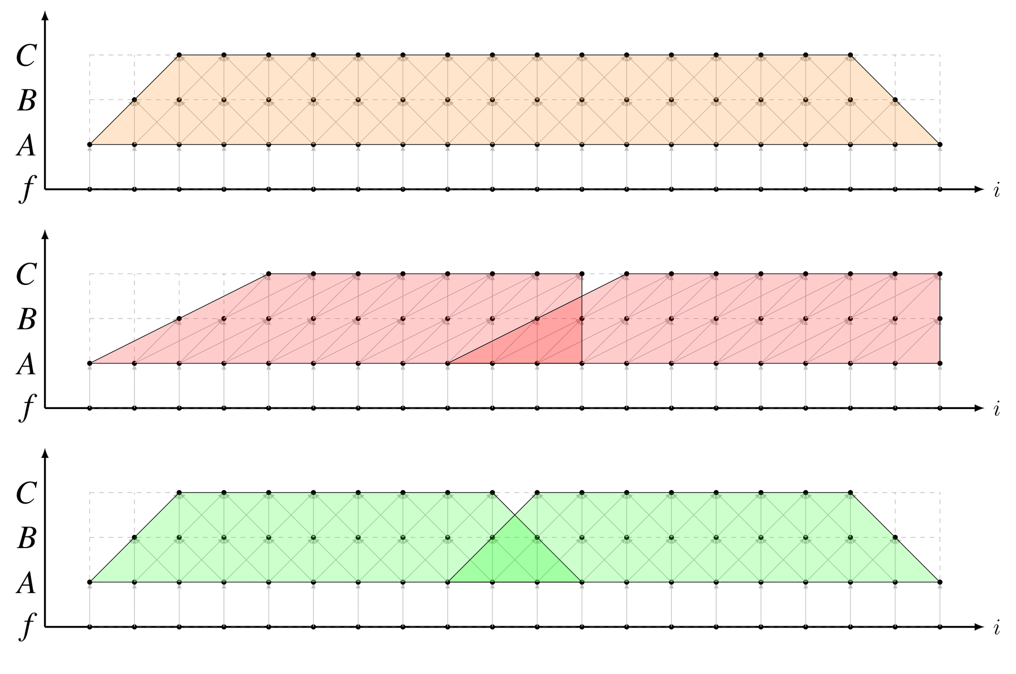

Figure 2: (a) Iteration domain; (b) overlapped tiling with shifting; (c) overlapped tiling without shifting

Polyhedral Implementation

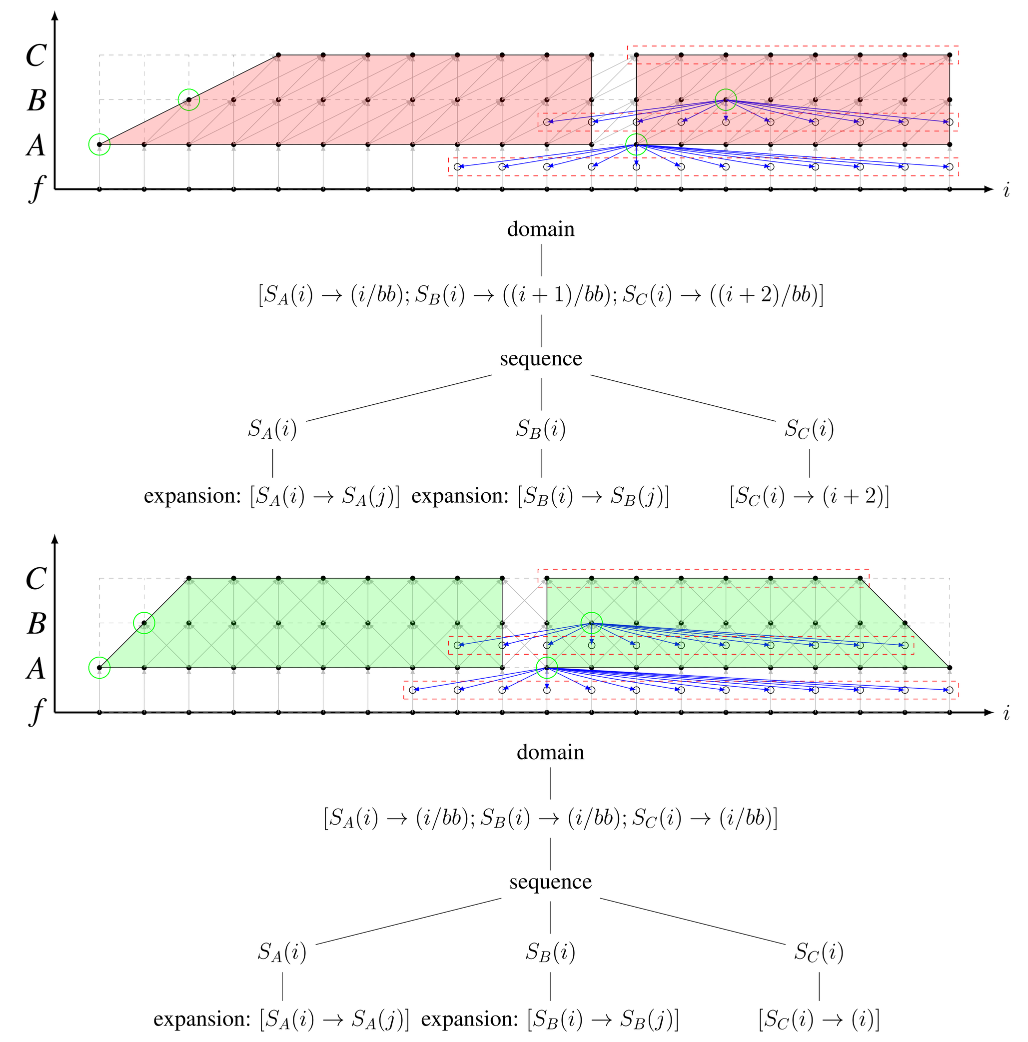

Figure 3: Implementing overlapped with expansion nodes (a) with shifting, and its (b) schedule tree; (c) without shifting, and its (d) schedule tree

Other Transformations

- Alignment and scaling of stages.
- Grouping/Fusion.
- Storage mapping.
- Hybrid tiling.

Experimental Results

- Code generator: PPCG (version `ppcg-0.07-26-g236d559`).
- Architecture: 32-core Intel Xeon(R) E5-2683 v4 @2.10GHz
- Compilation: ICC`18.0` (*-O3 -xHost -qopenmp -ipo*)
- Baseline:
  - sequential PolyMage naive code (without tiling) [5]
  - sequential stencil computations
- Comparison:
  - Halide manual and automatic scheduling [6], PolyMage naive and optimized scheduling [5]
  - Parallelogram tiling [2] and diamond tiling [1]

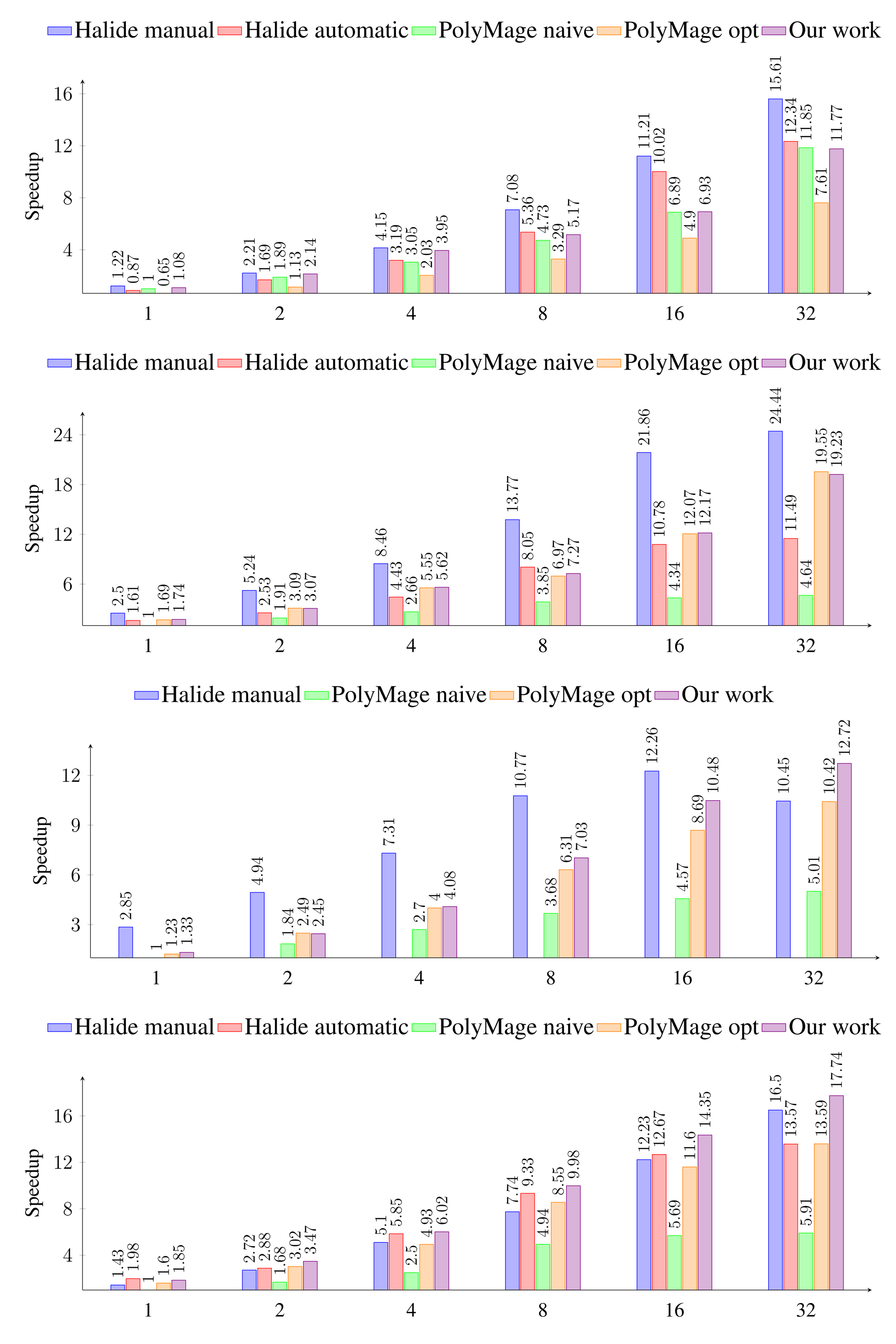

Figure 4: Performance Comparison of (a) Bilateral Grid; (b) Camera Pipelines; (c) Multiscale Interpolation; (d) Local Laplacian Filter

- Note: Halide automatic scheduling for Multiscale Interpolation is not publicly available.

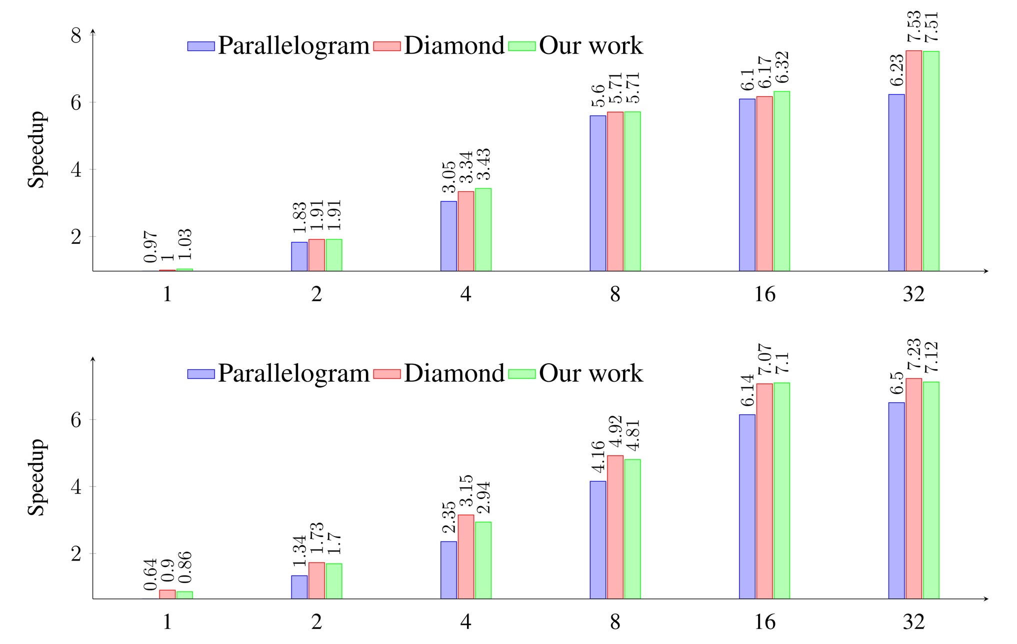

Figure 5: Performance Comparison of (a) heat-1d; (b) heat-2d

Conclusions

The polyhedral framework of compilation unifies a wide variety of loop and array transformations using affine (linear) transformations. The availability of a general purpose method to generate imperative code after the application of such affine transformations brought polyhedral compilers to the front scene. As a concurrent-startup-enabling tiling shape, overlapped tiling [4] is implemented in the domain specific code generator PolyMage [5]. In this work, we designed and implemented a general-purpose automatic overlapped tiling technique in a polyhedral framework by showing two implementations with/without shifting/skewing, followed by a series of experimental results on both image processing pipelines and stencil computations, validating the effectiveness of our method.

Forthcoming Research

We will be looking at all the remaining PolyMage benchmarks. Also, we have finished the CUDA code generation of these benchmarks. An in-depth comparison of all the PolyMage benchmarks by conducting experiments on both CPU and GPU architectures is coming soon.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant No. 61702546.

References

[1] Uday Bondhugula, Vinayaka Bandishti, and Irshad Pananilath. Diamond tiling: Tiling techniques to maximize parallelism for stencil computations. *IEEE Transactions on Parallel and Distributed Systems*, 28(5):1285–1298, October 2017.

[2] Uday Bondhugula, Albert Hartono, J. Ramanujam, and P. Sadayappan. A practical automatic polyhedral parallelizer and locality optimizer. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '08, pages 101–113, New York, NY, USA, 2008. ACM.

[3] F. Irigoin and R. Triolet. Supernode partitioning. In *Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '88, pages 319–329, New York, NY, USA, 1988. ACM.

[4] Sriram Krishnamoorthy, Muthu Baskaran, Uday Bondhugula, J. Ramanujam, Atanas Rountev, and P Sadayappan. Effective automatic parallelization of stencil computations. In *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '07, pages 235–244, New York, NY, USA, 2007. ACM.

[5] Ravi Teja Mullapudi, Vinay Vasista, and Uday Bondhugula. Polymage: Automatic optimization for image processing pipelines. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, pages 429–443, New York, NY, USA, 2015. ACM.

[6] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 519–530, New York, NY, USA, 2013. ACM.

[7] Sven Verdoolaege, Juan Carlos Juega, Albert Cohen, José Ignacio Gómez, Christian Tenllado, and Francky Catthoor. Polyhedral parallel code generation for cuda. *ACM Trans. Archit. Code Optim.*, 9(4):54:1–54:23, January 2013.