

Optimizing Code Generation for Matrix Multiplication

DAPHNE

Henri Willems

henri.willems@campus.tu-berlin.de

WS 23/24, TU Berlin

March 04, 2024

Matrix Multiplication Implementation

Matrix Multiplication is Level 3 Basic Linear Algebra Subprogram [\[1\]](#) operation

Hardware specific hand optimization, e.g. Intel oneMKL BLAS [\[2\]](#)

Daphne uses BLIS [\[3\]](#) a “portable software framework for instantiating high-performance BLAS-like dense linear algebra libraries”

[\[1\]](#) Dongarra, Cruz, Hammarling, *et al.* (1990). “Algorithm 679: A set of level 3 basic linear algebra subprograms: model implementation and test programs.”

[\[2\]](#) Intel® (). *Accelerate Fast Math with Intel® oneAPI Math Kernel Library.*

[\[3\]](#) Van Zee and van de Geijn (2015). “BLIS: A Framework for Rapidly Instantiating BLAS Functionality.”

[\[4\]](#) Lattner, Amini, Bondhugula, *et al.* (2021). “MLIR: Scaling Compiler Infrastructure for Domain Specific Computation.”

Matrix Multiplication Implementation

Matrix Multiplication is Level 3 Basic Linear Algebra Subprogram [1] operation

Hardware specific hand optimization, e.g. Intel oneMKL BLAS [2]

Daphne uses BLIS [3] a “portable software framework for instantiating high-performance BLAS-like dense linear algebra libraries”

Alternative: Automatic *Code Generation* using MLIR [4]

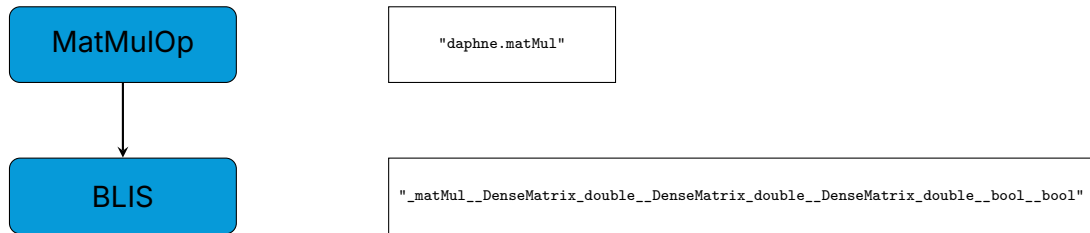
[1] Dongarra, Cruz, Hammarling, *et al.* (1990). “Algorithm 679: A set of level 3 basic linear algebra subprograms: model implementation and test programs.”

[2] Intel® (). *Accelerate Fast Math with Intel® oneAPI Math Kernel Library.*

[3] Van Zee and van de Geijn (2015). “BLIS: A Framework for Rapidly Instantiating BLAS Functionality.”

[4] Lattner, Amini, Bondhugula, *et al.* (2021). “MLIR: Scaling Compiler Infrastructure for Domain Specific Computation.”

Daphne Matrix Multiplication – Kernel



Daphne Matrix Multiplication – Code Generation

Project outcomes:

- Enabled Matrix Multiplication for `ui64` value type
- Extended code generation to `f32` and `si64`, `ui64` value types

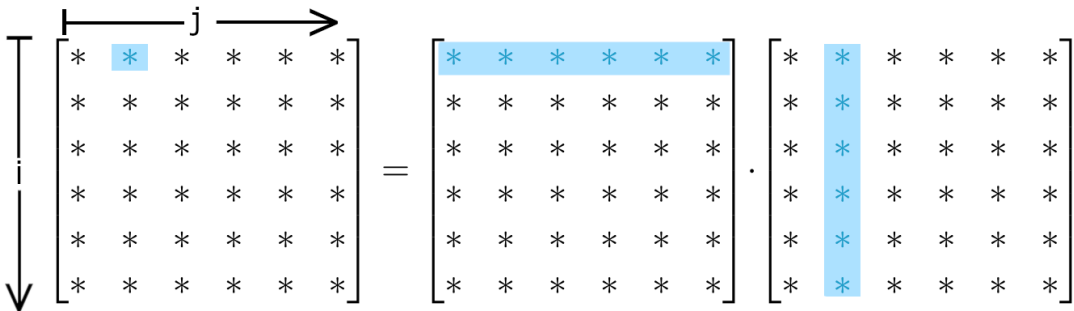
Daphne Matrix Multiplication – Code Generation

Project outcomes:

- Enabled Matrix Multiplication for `ui64` value type
- Extended code generation to `f32` and `si64`, `ui64` value types
- Enabled multiple Optimization strategies

Uday Bondhugula, "Using MLIR for High-Performance Code Gen: Part I," `bondhugula/llvm-project`. (Mar. 2, 2020), [Online]. Available: <https://github.com/bondhugula/llvm-project/blob/hop/mlir/docs/HighPerfCodeGen.md> (visited on 03/01/2024)

Matrix Multiplication



Matrix Multiplication – Inverted Loops

$$\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix} = \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix} \cdot \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}$$

k

Matrix Multiplication – Tiles and Vectors

The diagram illustrates matrix multiplication using tiles and vectors. It shows three 6x6 matrices arranged in an equation: Matrix A multiplied by Matrix B equals Matrix C.

Matrix A (left): A 6x6 matrix where the element at row 1, column 2 is highlighted with a red circle and a light blue background, representing a 1x1 tile.

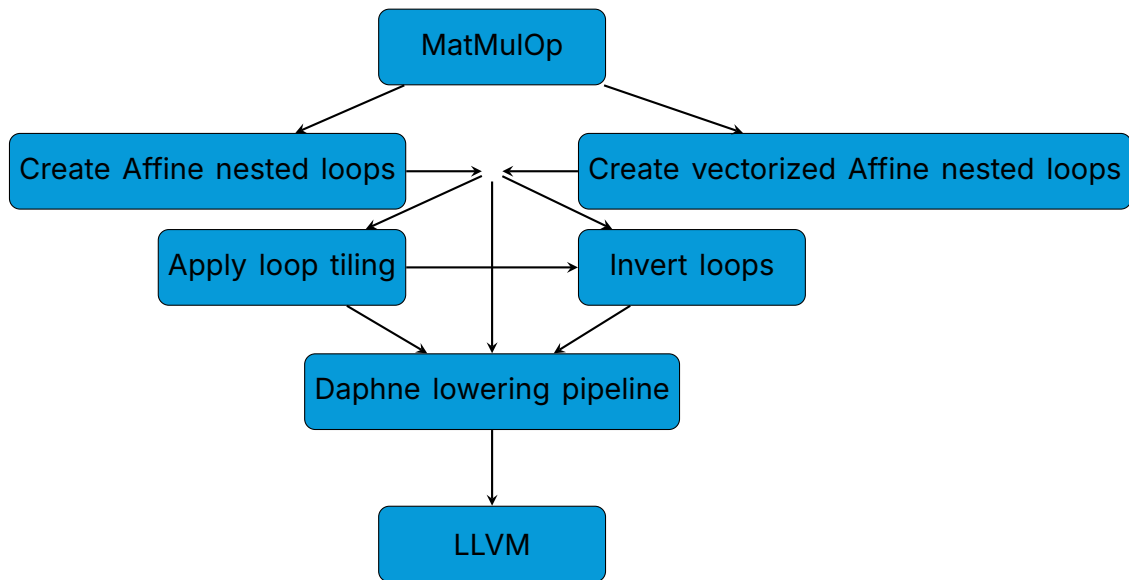
Matrix B (middle): A 6x6 matrix where a 2x3 block of elements (rows 1-2, columns 1-3) is highlighted with a red oval and a light blue background, representing a 2x3 tile.

Matrix C (right): A 6x6 matrix where a 2x1 block of elements (rows 1-2, column 2) is highlighted with a red oval and a light blue background, representing a 2x1 tile.

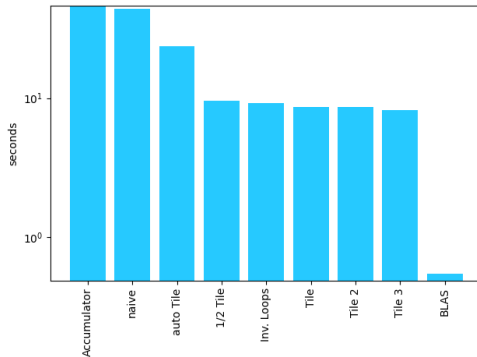
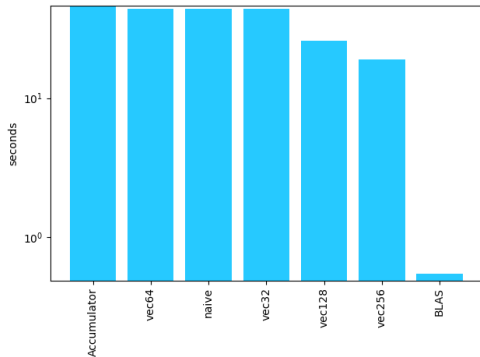
The equation is represented as:

$$\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix} = \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix} \cdot \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}$$

Daphne Matrix Multiplication – Code Generation

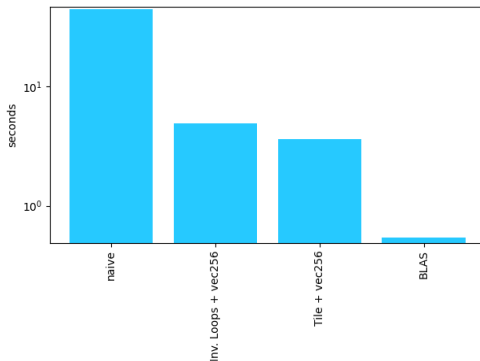


Optimizations enabled



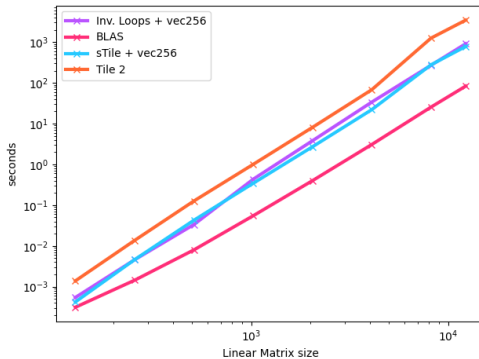
Execution times of dense 2048×2048 double precision Matrix multiplication with different vector instruction sizes (left) and tiling strategies (right) enabled.

Optimizations enabled – Tiling and Vectorization



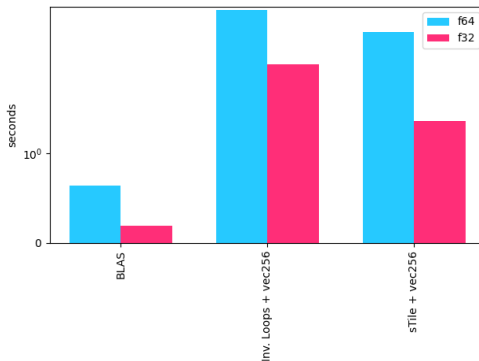
Execution times of dense 2048×2048 double precision Matrix multiplication with different optimizations.

Scaling behaviour



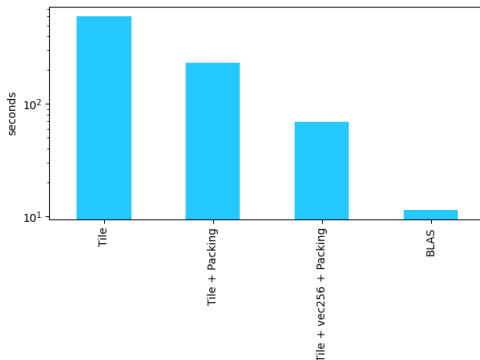
Execution times of dense double precision Matrix multiplications.

Single precision Matrix Multiplication



Execution times of 2048×2048 dense single and double precision Matrix multiplications.

Further improvement – Packing



Effect of packing, enabled through `affineDataCopyGeneratePass`, on 8192×8192 single precision Matrix multiplication executed with `mlir-cpu-runner`.

Bibliography

- [1] J. J. Dongarra, J. D. Cruz, S. Hammarling, and I. S. Duff, "Algorithm 679: A set of level 3 basic linear algebra subprograms: Model implementation and test programs," *ACM Trans. Math. Softw.*, vol. 16, no. 1, pp. 18–28, Mar. 1990, issn: 0098-3500. doi: 10.1145/77626.77627. [Online]. Available: <https://doi.org/10.1145/77626.77627>.
- [2] Intel®, "Accelerate Fast Math with Intel® oneAPI Math Kernel Library," Intel.com. (), [Online]. Available: <https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html> (visited on 03/01/2024).
- [3] F. G. Van Zee and R. A. van de Geijn, "BLIS: A framework for rapidly instantiating BLAS functionality," *ACM Transactions on Mathematical Software*, vol. 41, no. 3, 14:1–14:33, Jun. 2015. [Online]. Available: <https://doi.acm.org/10.1145/2764454>.

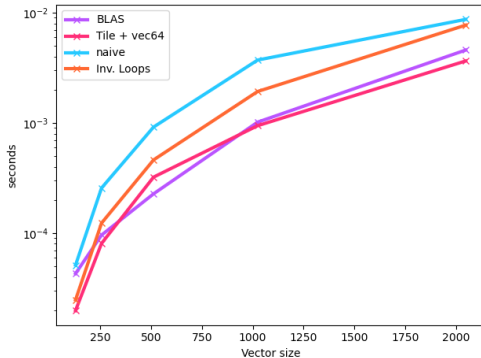
Bibliography

- [4] C. Lattner, M. Amini, U. Bondhugula, *et al.*, "MLIR: Scaling compiler infrastructure for domain specific computation," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021, pp. 2–14. doi: 10.1109/CGO51591.2021.9370308.
- [5] Uday Bondhugula, "Using MLIR for High-Performance Code Gen: Part I," bondhugula/llvm-project. (Mar. 2, 2020), [Online]. Available: <https://github.com/bondhugula/llvm-project/blob/hop/mlir/docs/HighPerfCodeGen.md> (visited on 03/01/2024).

Henri Willems

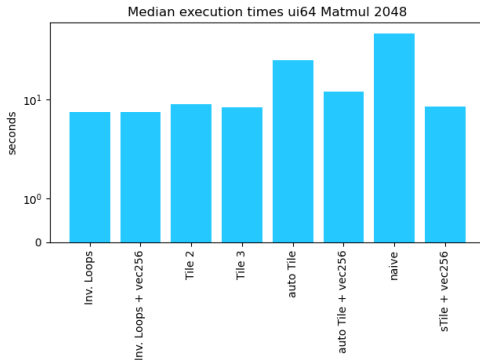
henri.willems@campus.tu-berlin.de

Appendix



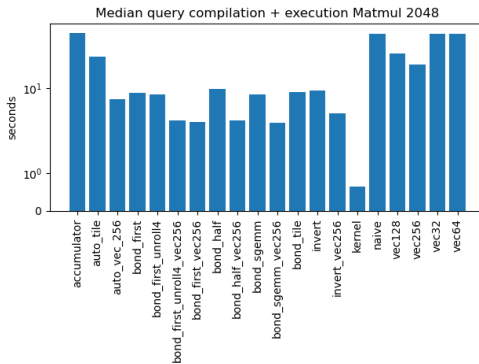
Execution times of double precision Matrix Vector product.

Appendix

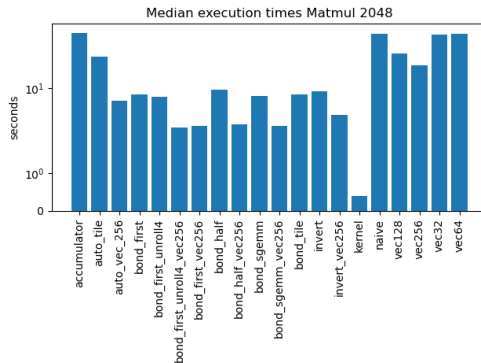


Execution times of ui64 2048×2048 Matrix multiplication. Note this is not supported by the provided Kernel implementation.

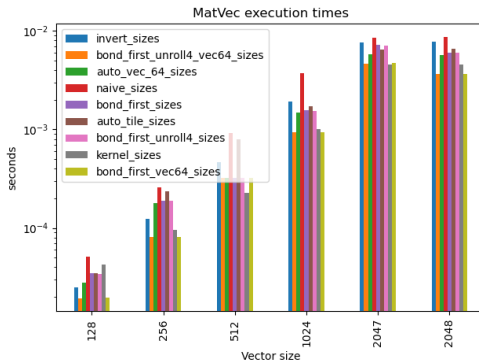
Appendix



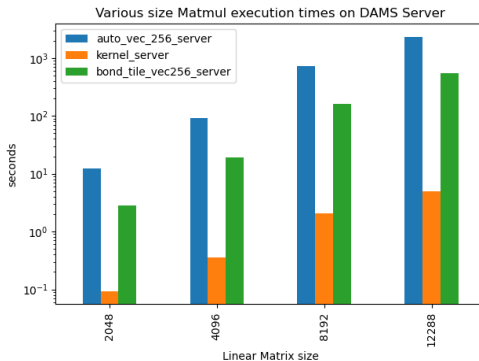
Appendix



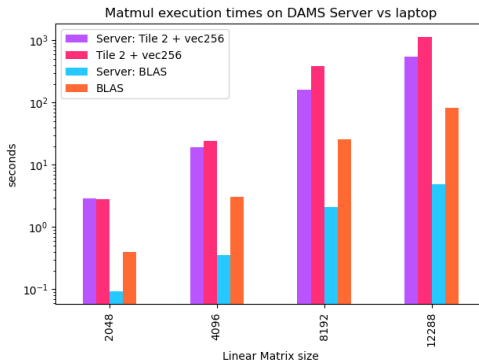
Appendix



Appendix



Appendix



Appendix

