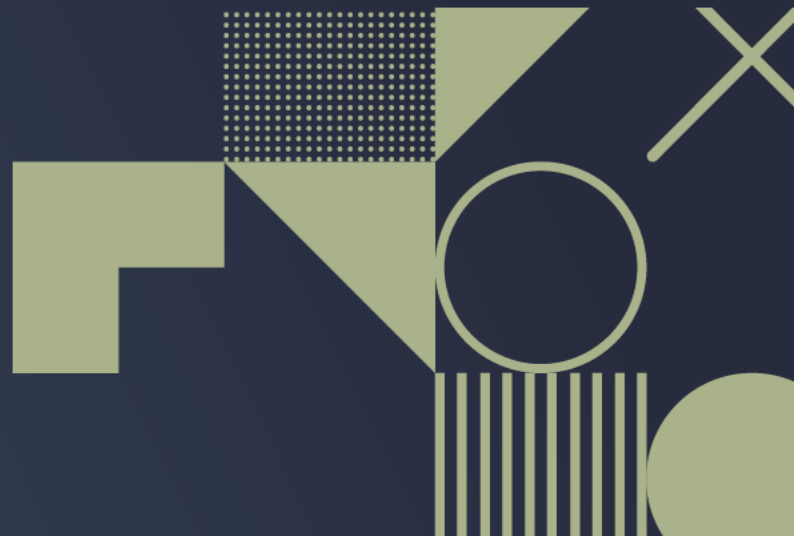

# Reducing Redundancy in Data Organization and Arithmetic Calculation for Stencil Computations

Kun Li, Liang Yuan, Yunquan Zhang, Yue Yue

State Key Laboratory of Computer Architecture,
Institute of Computing Technology, Chinese Academy of Sciences



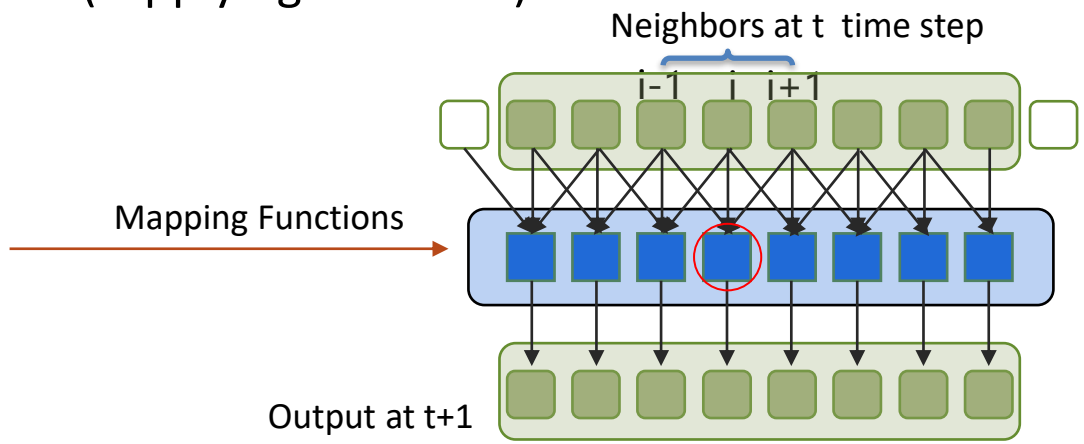
# Outline

- **Background and Motivation**
- Spatial Computation Folding
- Temporal Computation Folding
- Optimization
- Summary

# Background: What is the stencil?

- Stencil computation is one of the most important kernels in various scientific and engineering applications.
- For a given point, a stencil is a pre-determined set of nearest neighbors (possibly including itself).
- A stencil pattern updates every point in a regular grid depends on a mapping function that accesses a weighted subset of its neighbors ("applying a stencil").

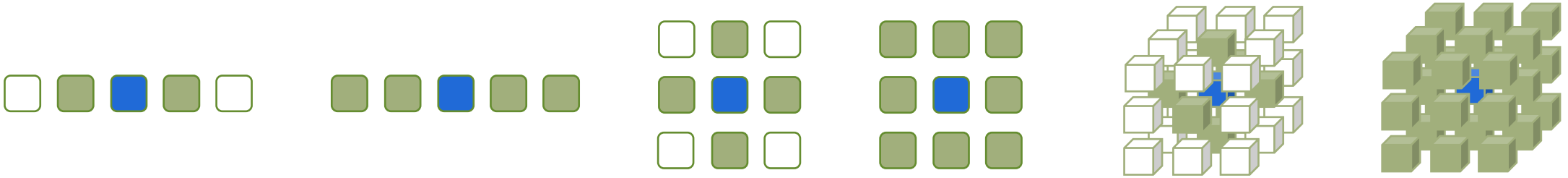
```
for (i = 1; i < N; ++i)
   B[i] = a * (A[i-1] + A[i] + A[i+1]);
endfor
```




Jacobi 1D3P Stencil Kernel

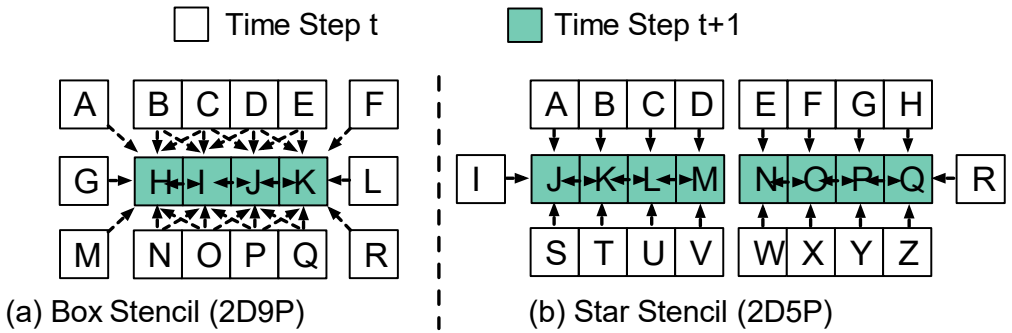
# Background: Stencil Applications

- Stencils are critical to many scientific applications:
  - Diffusion, Electromagnetics, Computational Fluid Dynamics
  - Both explicit and implicit iterative methods (e.g. Multigrid)
  - Both uniform and adaptive block-structured meshes
- Many type of stencils
  - 1D, 2D, 3D meshes
  - Number of neighbors (5-pt, 7-pt, 9-pt, 27-pt,…)
  - Gauss-Seidel (update in place) vs Jacobi iterations (2 meshes)



# Motivation: Data Alignment Conflict

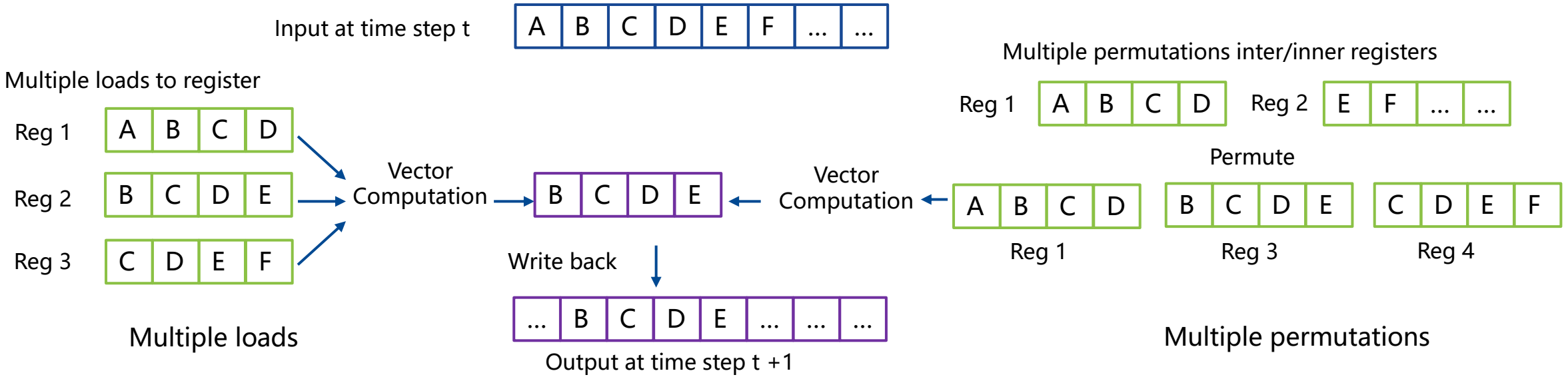
- Data alignment conflict is a problem caused by vectorization, where the neighbors for a grid point appear in the same vector register but at different positions.
  - Redundant references and unaligned vector loads in one single vector computation.
  - Poor data reuse between continuous computations.
  - Constrained to limited registers, disadvantages worsen as the order or dimensionality of stencil increases.




Spatial data conflicts in vectorization of 2D stencils

# Motivation : Data Alignment Conflict

- To address the problem of spatial conflicts, two common implementations are often adopted.
  - Multiple loads: easy to achieve, while increase the data transfer volume and unaligned memory references.
  - Multiple permutations: reduce the memory bandwidth usage, while redundant data preparation.



# Outline

- Motivation and Background
- Spatial Computation Folding
  - **Scalar Folding**
  - Vectorization
- Temporal Computation Folding
- Optimization
- Summary

# Scalar Folding

- Calculate the stencil along outer spatial dimensions first and perform the unit-stride loop computation after a data layout transformation that eliminates the data conflicts.

- Calculation along one dimension defined as a *folding*.

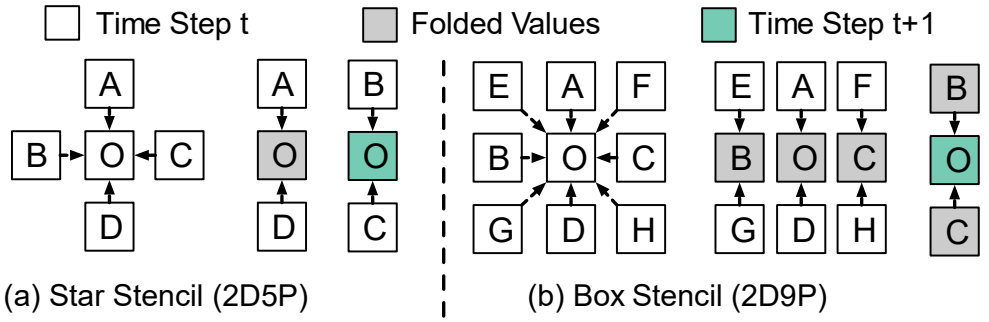



Illustration of computation folding strategy for scalar 2D stencils.

# Outline

- Motivation and Background
- Spatial Computation Folding
  - Scalar Folding
  - **Vectorization**
- Temporal Computation Folding
- Optimization
- Summary

# Vectorization

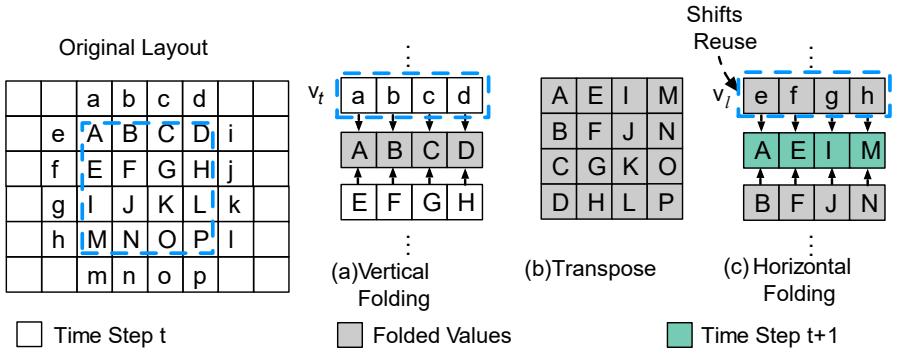
- Group vectors as a vector set, which is the basic processing granularity.
- Characterized by redundancy-free vertically, a set of vertical folding is performed first on a vector set.
- In-register transpose for horizontal folding.
- The innermost loop pipelines the stencil computation.

# Example

- Vector Set $VS_1$ is loaded into registers.

- The top and bottom vectors for this vector set are loaded to $v_t$ = ($a$, $b$, $c$, $d$) and $v_b$ = ($m$, $n$, $o$, $p$) respectively and then for a vertical folding.

- Horizontal folding is performed to gather the folded values based on the transposed layout in registers.




Vectorized process for 2D9P box stencil.

# Outline

- Motivation and Background
- Spatial Computation Folding
- Temporal Computation Folding
  - **Motivation**
  - Scalar Profitability Analysis
  - Vectorized Multi-step Computation
- Optimization
- Summary

# Motivation

- Most of the existing work utilizes blocking technique to decrease the data transfers between main memory and cache, there is no in-register data reuse between successive time loops.

- Straightforward implementation of reusing registers along the time dimension easily exacerbates excessive register spilling.

- Our approach is to seek a balance that the redundancy of arithmetic calculation is eliminated along the time dimension, and register pressure is alleviated simultaneously.

# Outline

- Motivation and Background
- Spatial Computation Folding
- Temporal Computation Folding
  - Motivation
  - **Scalar Profitability Analysis**
  - Vectorized Multi-step Computation
- Optimization
- Summary

# Scalar Profitability Analysis

- An arithmetic collect $C(E)$ is defined to describe the number of arithmetical instructions (add, multiply, multiply-add, etc.) used for 2-step updates in the stencil expression $E$.

$$C(E) = \bigcup_{s} \{\langle g, w_g \rangle | \langle g, w_g \rangle \in C(E_s)\}$$

- The new collect $C(E_\Lambda)$ could be obtained from the temporal computation folding on this point set with each grid point folded by $\lambda g$.

$$C(\mathbb{E}_\Lambda) = \{\langle g, \lambda_g \rangle | \text{ grid } g \text{ used in } \mathbb{E}_\Lambda \text{ weighted with } \lambda_g\}$$

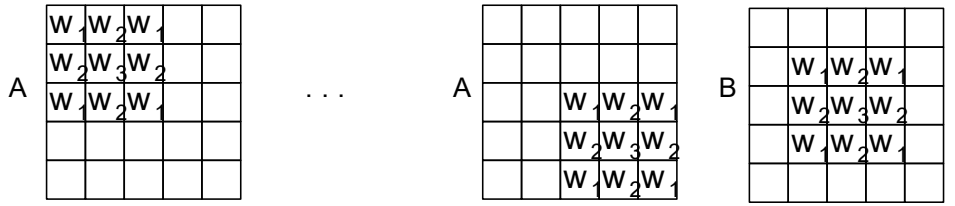
- The profitable index is defined below, and a profitable folding means the fraction of the cardinalities on two sets at least exceeds a threshold $\theta \geq 1$.

$$P(E, \mathbb{E}_\Lambda) = \frac{|C(E)|}{|C(\mathbb{E}_\Lambda)|} \geq \theta$$

# Example

- In each subexpression $E_s$ in 2D9P stencil, a pre-defined weight $w_g$ is assigned on each grid point and then a 9-way addition result is obtained for $|C(E)| = 10 \times |C(E_s)| = 90$.

- It is worth noting that redundant arithmetic operations are performed iteratively on the same point in different subexpressions, and store/reload operations incur additional interrupts during the computation process.




```
for (i = -1; i ≤ 1; ++i)
    for (j = -1; j ≤ 1; ++j)
        B[i,j] = w1*(A[i-1,j+1]+A[i+1,j+1]+A[i-1,j-1]+A[i+1,j-1])+
                 w2*(A[i-1,j]+A[i+1,j]+A[i,j-1]+A[i,j+1])+
                 w3*A[i,j]
    endfor
endfor
```

|C(E_s)|=9

---------- INTERRUPTS OF STORE/RELOAD OPERATIONS ----------

```
A[0,0] = w1*(B[-1,1]+B[1,1]+B[-1,-1]+B[1,-1])+
         w2*(B[-1,0]+B[1,0]+B[0,-1]+B[0,1])+
         w3*B[0,0]
```

|C(E)|=90

Illustration of scalar arithmetic expression for updating 2 time steps on the center point A(0,0) by baseline approach.

# Example

- A new arithmetical expression $E_\Lambda$ is determined by the folding matrix comprised of new weights $\lambda$.
- The five associative grid points of the same column are folded with $\lambda$ first, and then a Horizontal Folding is performed to gather the obtained five folded values.
- The new arithmetic collect is 25, and it gives a net profitable index of $P$ ($E$, $E_\Lambda$) = 90/25 = 3.6.
- Moreover, the interrupt cost of store/reload operations is also eliminated completely in $E_\lambda$ .

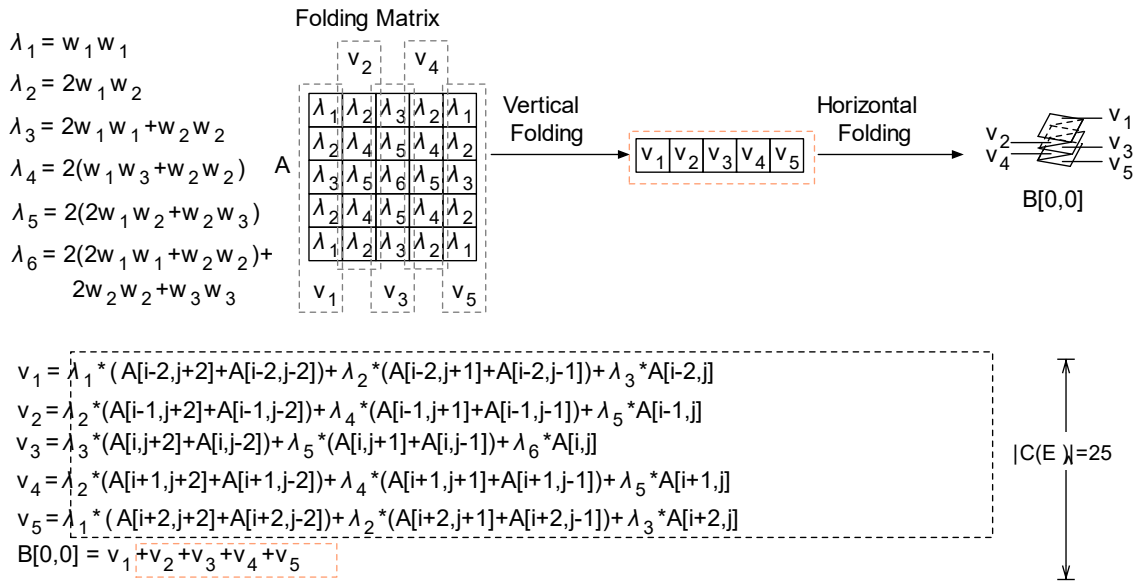



Illustration of scalar arithmetic expression for updating 2 time steps on the center point A(0,0) by our approach.

# Outline

- Motivation and Background
- Spatial Computation Folding
- Temporal Computation Folding
  - Motivation
  - Scalar Profitability Analysis
  - **Vectorized Multi-step Computation**
- Optimization
- Summary

# Vectorized Multi-step Computation

- Data Preparation: load basic granularity as a vector set.
- Vertical Folding: collect neighbor points in the same column.
- Horizontal Folding: a local transpose is performed to collect the folded values in the same row.
- Weighted Transpose: followed by a weighted transpose at last.

# Example

- Take a 2-step 2D9P box stencil as an example.

2D9P Box Codes

```
for (t = 1; t ≤ T; ++t)
    for (i = 1; i ≤ N; ++i)
        for (j = 1; j ≤ N; ++j)
            B[i,j] = w*(A[i-1,j+1]+A[i+1,j+1]+A[i-1,j-1]+
                        A[i+1,j-1]+A[i-1,j]+A[i+1,j]+
                        A[i,j-1]+A[i,j+1]+A[i,j])
        endfor
    endfor
endfor
```

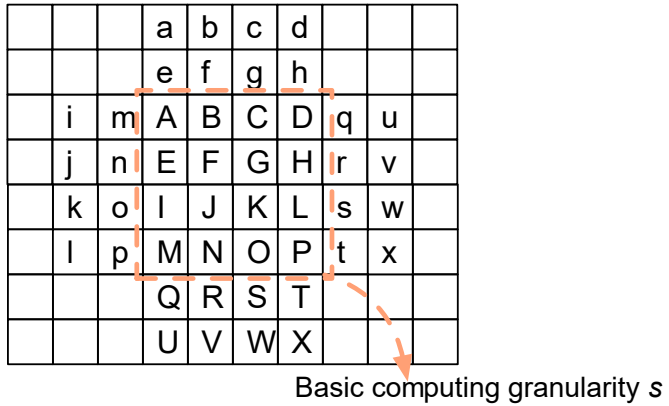
# Example

- Data Preparation.
  - Compute the folding matrix comprised of new weights.
  - The basic granularity of temporal folding is also constructed as a 4×4 square of grid points.
  - They are loaded from cache to four registers at time step $t$.

Folding Matrix

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 2 | 1 |
| 2 | 4 | 6 | 4 | 2 |
| 3 | 6 | 9 | 6 | 3 |
| 2 | 4 | 6 | 4 | 2 |
| 1 | 2 | 3 | 2 | 1 |

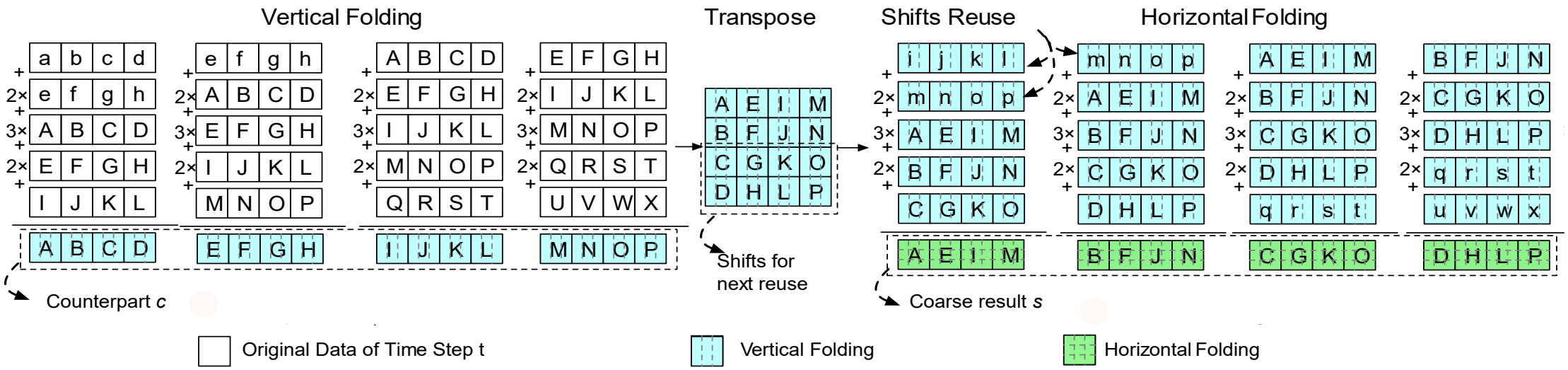
Original Layout



# Example

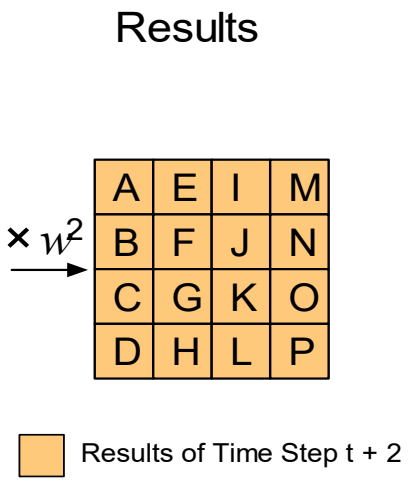
- Vertical Folding.
  - The weights for the first counterpart are reassigned as $\boldsymbol{\lambda}$ ={1, 2, 3, 2, 1} by the folding matrix, i.e., each $v_i$ in the first counterpart is calculated by performing a sum on $v_{i-2}$, $2v_{i-1}$, $3v_i$ , $2v_{i+1}$, and $v_{i+2}$.

- Horizontal Folding.
  - A local transpose is performed in registers.
  - Compute by reassigned weights.



# Example

- Weighted Transpose.
  - Horizontal folding is followed by a weighted transpose at last.
  - The local transpose can be also optimized away here.

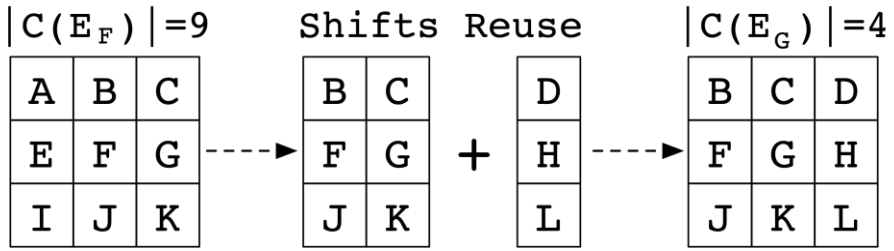


# Outline

- Motivation and Background
- Spatial Computation Folding
- Temporal Computation Folding
- Optimization
  - **Shifts Reusing**
  - Tessellate Tiling
  - Code Generation
- Summary

# Shifts Reusing

- Avoid redundant arithmetical calculations and repetitive data transfers by utilizing the same data collected in the last round as input to be computed together.
  - A brief sketch of the scalar 1-step stencil computations moving from adjacent grid points F to G in data space.
  - There is potential for reusing shifts within the successive stencil computation from grid point F to G, which gives another reuse profitability of 2.25.



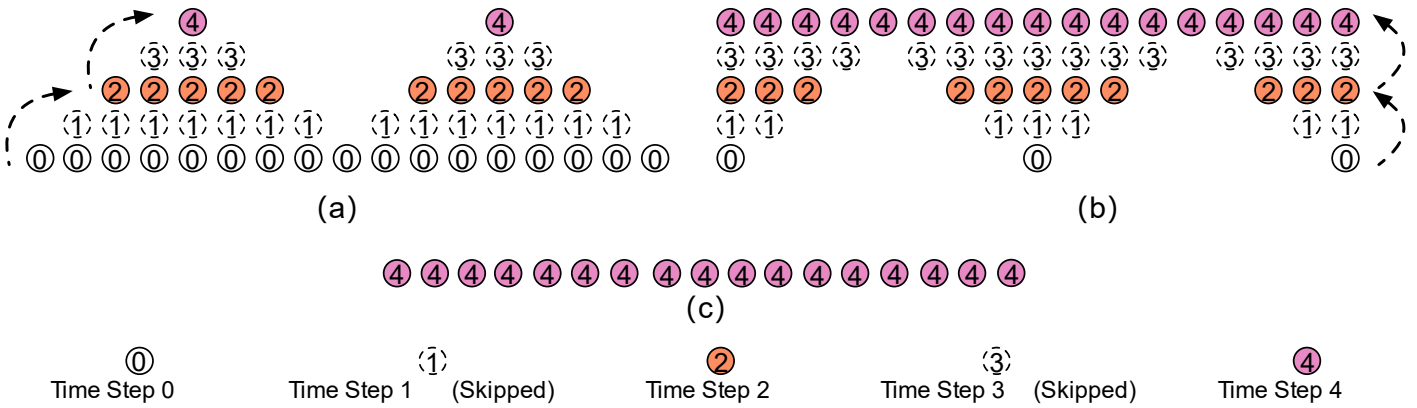
# Outline

- Motivation and Background
- Spatial Computation Folding
- Temporal Computation Folding
- Optimization
  - Shifts Reusing
  - **Tessellate Tiling**
  - **Code Generation**
- Summary

# Tessellate Tiling & Code Generation

- Tiling serves to exploit the data reuse at cache levels, while vectorization boosts the computation using the data parallelism at the execution level.
  - Use the tessellation framework to implement fine integration with vectorized approaches.
  - Exploit the data reuse at cache levels.

- A generator serves to reduce the code size and achieve performance portability on different architectures.
  - We abstract the algorithmic skeletons of the outer loops as a semi-automatic code generator.
  - The output of the code generator is integrated into C code to simplify the parallel programming of stencils.




Tessellate tiling integration.

# Outline

- Motivation and Background
- Spatial Computation Folding
- Temporal Computation Folding
- Optimization
- Summary
  - **Analytical register behaviors**
  - Sequential Block-free Results
  - Scalability
  - Impact of Data Preparation
  - Multicore Cache-blocking Experiments
  - Discussion and Conclusion

# Analytical register behaviors

- The Table lists the numbers of arithmetic, load, store and data reorganization operations.
- Compared with the auto vectorization and data reorganization methods, spatial folding achieves fewer data accesses for all stencils and fewer computation instructions for higher-dimensional box stencils.

| Kernel | 1D5P | | | | 2D9P | | | | 3D27P | | | | 1D-Heat | | | | 2D-Heat | | | | 3D-Heat | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Operation[1] | C. | L. | S. | P. | C. | L. | S. | P. | C. | L. | S. | P. | C. | L. | S. | P. | C. | L. | S. | P. | C. | L. | S. | P. |
| AutoVec.[2] | 5 | 5 | 1 | 0 | 9 | 9 | 1 | 0 | 27 | 27 | 1 | 0 | 3 | 3 | 1 | 0 | 5 | 5 | 1 | 0 | 7 | 7 | 1 | 0 |
| Reorg. | 5 | 1 | 1 | 4 | 9 | 3 | 1 | 6 | 27 | 9 | 1 | 18 | 3 | 1 | 1 | 2 | 5 | 3 | 1 | 2 | 7 | 5 | 1 | 2 |
| S-Fold | 5 | 1 | 1 | 0 | 5 | 1.5 | 1 | 1 | 15 | 4.5 | 1 | 3 | 3 | 1 | 1 | 0 | 5 | 1.5 | 1 | 1 | 7 | 2 | 1 | 1 |
| S&T-Fold | 4.5 | 0.5 | 0.5 | 0 | 5 | 1 | 0.5 | 0.5 | 10 | 2.5 | 0.5 | 1.5 | 2.5 | 0.5 | 0.5 | 0 | 5 | 1 | 0.5 | 1 | 10 | 1.5 | 0.5 | 2 |

[1] For better clarity, Computation, Load, Store, and Permute operations are abbreviated with C., L., S., and P. respectively.
[2] Methods for Auto Vectorization, Data Reorganization, Spatial Folding, and Spatial&Temporal Folding are also abbreviated accordingly (similarly hereinafter).

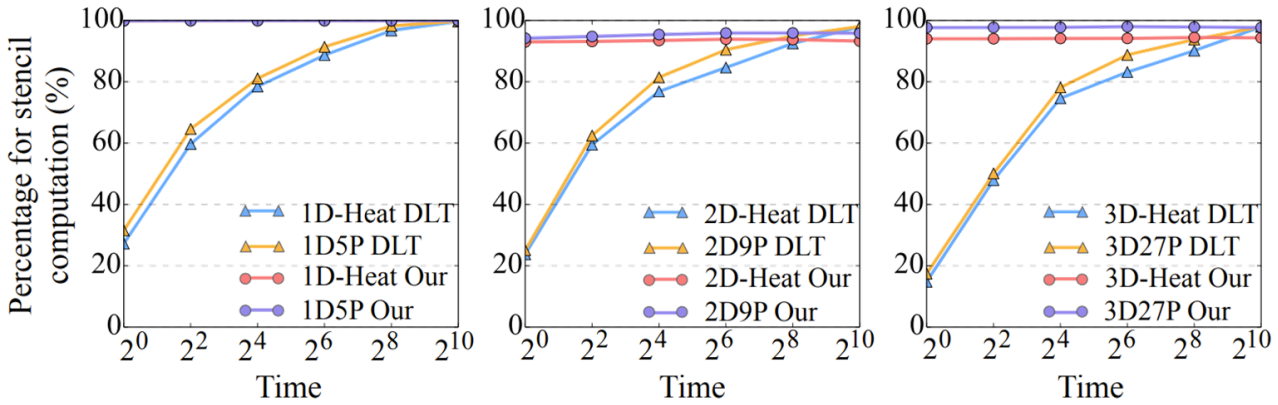
Analytical register behaviours for different methods on Jacobi stencils (per vector)

# Outline

- Motivation and Background
- Spatial Computation Folding
- Temporal Computation Folding
- Optimization
- Summary
  - Analytical register behaviors
  - Sequential Block-free Results
  - Scalability
  - **Impact of Data Preparation**
  - Multicore Cache-blocking Experiments
  - Discussion and Conclusion

# Impact of Data Preparation

- DLT method is sensitive to dimension and time size.
  - The number of time loops should be large enough to amortize the data preparation overhead.
  - An additional array is also required to store the transposed data by DLT, which increases the storage pressure.
- Our method could obtain a high computing density continuously.
  - The in-register transpose makes little difference on the overall performance.




The percentages of stencil computations for Jacobi stencils in single-thread blocking-free experiments.

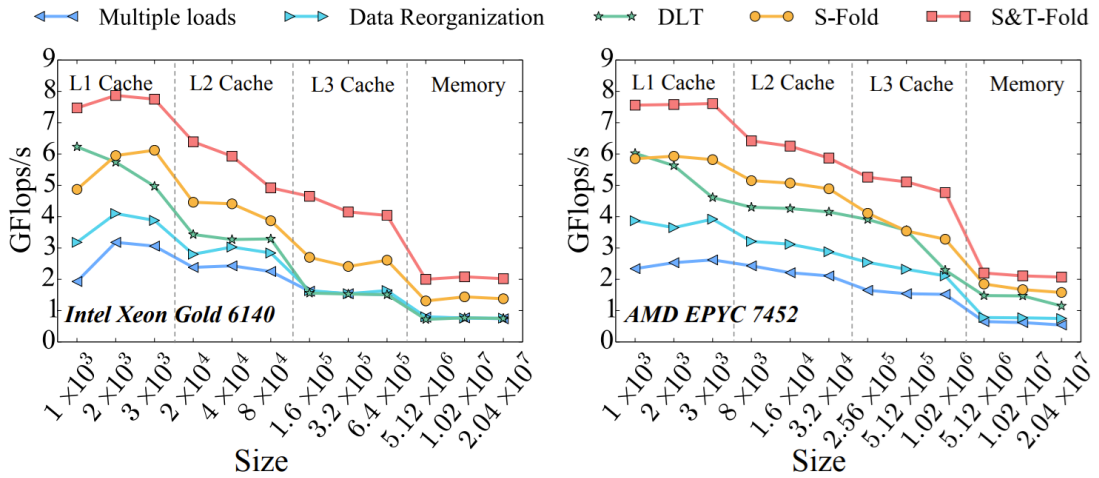
# Outline

- Motivation and Background
- Spatial Computation Folding
- Temporal Computation Folding
- Optimization
- Summary
  - Analytical register behaviors
  - **Sequential Block-free Results**
  - Scalability
  - Impact of Data Preparation
  - Multicore Cache-blocking Experiments
  - Discussion and Conclusion

# Sequential Block-free Results

- Present the performance results of varied methods across problem sizes ranging from L1 cache to main memory on a single thread without cache-blocking technique by two machines.

- Our method outperforms others apparently in both machines.
  - The multiple loads method exhibits the worst performance among them due to the overhead caused by redundant loads.
  - The performance drops apparently as the problem size moves from L1 cache to the memory hierarchy, which is mainly caused by the cost of data transfers.




Absolute performance comparison for tested methods in single-thread blocking-free experiments (Intel/AMD).

# Outline

- Motivation and Background
- Spatial Computation Folding
- Temporal Computation Folding
- Optimization
- Summary
  - Analytical register behaviors
  - Sequential Block-free Results
  - Scalability
  - Impact of Data Preparation
  - **Multicore Cache-blocking Experiments**
  - Discussion and Conclusion

# Multicore Cache-blocking Experiments

- Present the experiments that exhibit the benefits of our methods with cache-blocking techniques and parallelization scheme.

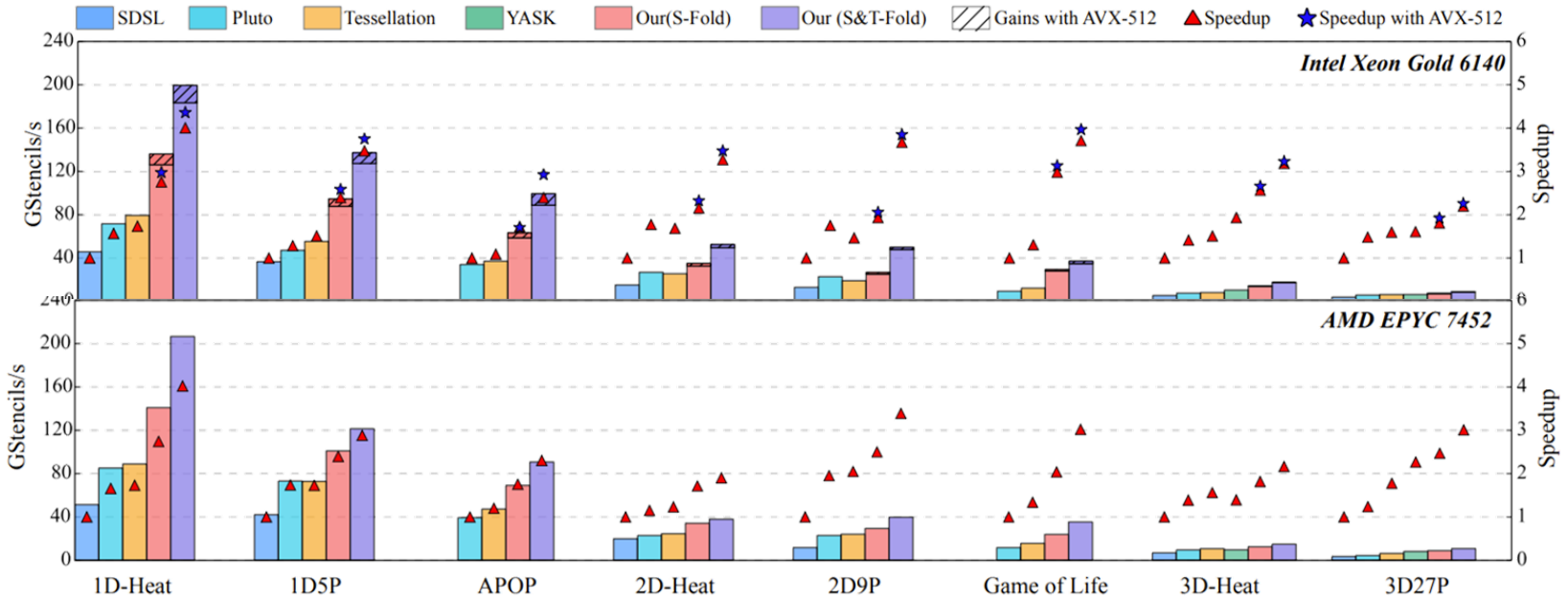
- Combine our vectorization scheme with tessellate tiling, and compare with the SDSL, Pluto, YASK and Tessellation.

| Benchmarks | vectorization | Blocking | Parallelization |
|---|---|---|---|
| SDSL | DLT | Split tiling | OpenMP |
| Pluto | AutoVec. | Diamond tiling | OpenMP |
| Tesselation | AutoVec. | Tessellate tiling | OpenMP |
| YASK | Vector Folding | Loop tiling | OpenMP |
| Our | S&T-Fold | Tessellate tiling | OpenMP |

Techniques for vectorization, cache-blocking, and parallelization in benchmarks

# Multicore Cache-blocking Experiments

- Our vectorization scheme provides a significant benefit in a large problem size compared to the referenced work.
  - Taking all stencils with AVX2 instructions into account, remarkable performance improvements are observed from our method with S&T-Fold.
  - The optimization with AVX-512 instructions could obtain further performance gains.



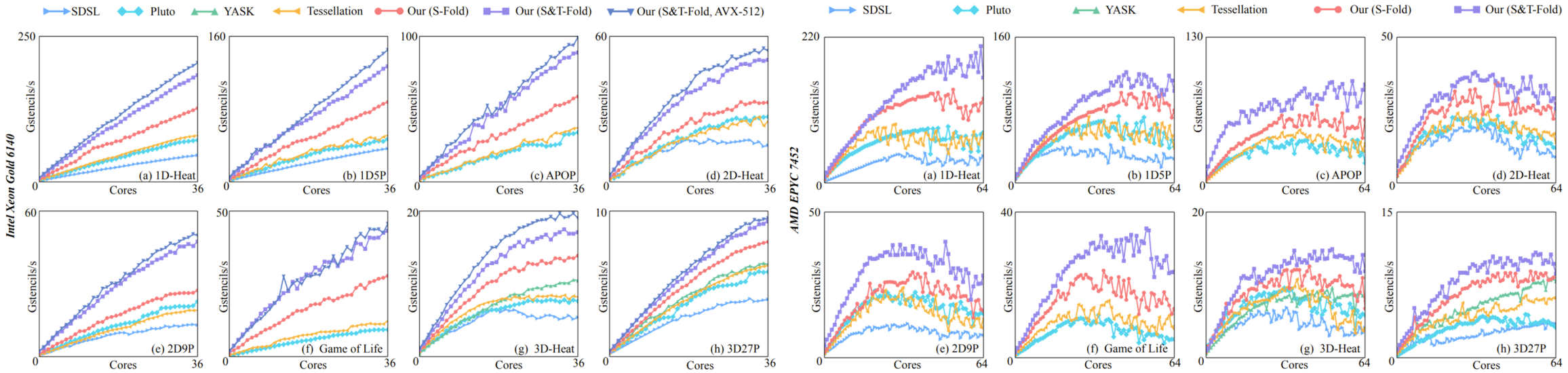

Performance and speedup comparison with cache-blocking on multicore Intel/AMD machine.

# Outline

- Motivation and Background
- Spatial Computation Folding
- Temporal Computation Folding
- Optimization
- Summary
  - Analytical register behaviors
  - Sequential Block-free Results
  - **Scalability**
  - Impact of Data Preparation
  - Multicore Cache-blocking Experiments
  - Discussion and Conclusion

# Scalability

- Our method could achieve the highest performance while the SDSL obtain the lowest performance.
  - In 1D-Heat stencils, all these methods achieve nearly linear scaling on both instruction sets.
  - The scalability for all methods drops as a result of the inherent complexity for multidimensional stencil computations.
  - The performance of AVX-512 optimization shows a further increase.




Scalability for stencils of various orders with different dimensions on multicore Intel/AMD machine.

# Outline

- Motivation and Background
- Spatial Computation Folding
- Temporal Computation Folding
- Optimization
- Summary
  - Analytical register behaviors
  - Sequential Block-free Results
  - Scalability
  - Impact of Data Preparation
  - Multicore Cache-blocking Experiments
  - **Conclusion**

# Discussion

- Impacts of Data Preparation
  - High performance of a state-of-the-art vectorized method DLT is obtained by the cost of specific stencil parameters (low dimension or long time size).
- Sequential Block-free
  - Overall performance trends drop consistently with the memory hierarchy.
- Multicore Tiling
  - Frequency reduction called throttling exists in CPUs when heavy AVX2 and AVX-512 extensions are involved, and it leads to a mediocre performance in 3D stencils.
- Scalability
  - Our vectorized scheme leveraging tessellate tiling outperforms the referenced benchmarks across a broad variety of configurations.
  - More inter-core communication is a performance-limiting factor as cores increase.

# Outline

- Motivation and Background
- Spatial Computation Folding
- Temporal Computation Folding
- Optimization
- Summary
  - Analytical register behaviors
  - Sequential Block-free Results
  - Scalability
  - Impact of Data Preparation
  - Multicore Cache-blocking Experiments
  - **Conclusion**

# Conclusion

- We propose a novel *spatial computation folding* strategy to overcome the spatial data conflicts efficiently for vectorized stencils.

- Based upon the new proposed strategy, we design a *temporal computation folding* approach enhanced with *shifts reusing*, *tessellate tiling* and *semi-automatic code generation*. It aims to reduce the redundancy of arithmetic calculation in time iteration space.

- We *generalize* our approach on various kernels, and demonstrate that it could achieve superior performance compared to different highly-optimized work on Intel/AMD multi-core processors with AVX2/AVX-512 instruction sets.

# Q&A


# Reducing Redundancy in Data Organization and Arithmetic Calculation for Stencil Computations

Kun Li, Liang Yuan, Yunquan Zhang, Yue Yue

State Key Laboratory of Computer Architecture,
Institute of Computing Technology, Chinese Academy of Sciences