About Me - Muhong Zhou

- CAAM 2nd year graduate student
- B.S. in Math and Applied Math, Zhejiang University, July 2011

Using Time-Skewing Approach to Accelerate Stencil Codes for Acoustic Constant Density Wave Equation

Muhong Zhou

CAAM
Rice University

Overview

- Finite Difference Formulation for ACD Wave Equation
- Naive Stencil Implementation and Problem Description
- Time-Skewing Approach and Computational Analysis
- Results
- Conclusion

Finite Difference Formulation for ACD Wave Equation

- Acoustic Constant Density(ACD) Wave Equation:

$$\frac{1}{C^2} U_{tt}(t, \vec{X}) - \bigtriangledown^2_{\vec{X}} U(t, \vec{X}) = 0$$

- Assumption: velocity C is a constant, but in real world it is not!
- $t \in [0, T], \vec{X} \in [0, X] \times [0, Y] \times [0, Z]$
- Initial Condition $\frac{\partial U}{\partial t}(0, \vec{X})$ and $U(0, \vec{X})$.
- Boundary Condition

$$U(0, \vec{X}) = 0 \quad \text{if } x \in \{0, X\} \text{ or } y \in \{0, Y\} \text{ or } z \in \{0, Z\}$$

Finite Difference Formulation for ACD Wave Equation

- D_i^n: centered difference approximation for second order derivative
  i: direction, n: error order
  e.g. $D_t^2 U = \frac{U(t-\delta t,\vec{X})-2U(t,\vec{X})+U(t+\delta t,\vec{X})}{\delta t^2}$
- 2-4 Finite difference formulation for ACD wave equation is:

$$\frac{1}{C^2} D_t^2 U = D_x^4 U + D_y^4 U + D_z^4 U$$

Finite Difference Formulation for ACD Wave Equation

- D_i^n: centered difference approximation for second order derivative
  i: direction, n: error order
  e.g. $D_t^2 U = \frac{U(t-\delta t,\vec{X})-2U(t,\vec{X})+U(t+\delta t,\vec{X})}{\delta t^2}$
- 2-4 Finite difference formulation for ACD wave equation is:

$$\frac{1}{C^2} D_t^2 U = D_x^4 U + D_y^4 U + D_z^4 U$$

- $U(t+\delta t, \vec{X}) = -U(t-\delta t, \vec{x}) + 2 * U(t, \vec{x}) + C^2 * \delta t^2 (D_x^4 U + D_y^4 U + D_z^4 U)$
- Data values at three different time steps are involved, and need two data arrays to store these temporary data values

Naive Stencil Implementation and Problem Description

- Grid: $\delta x, \delta y, \delta z$ are unit distances on each direction
- `Up[i][j][k]` represents $U(t - \delta t, i * \delta x, j * \delta y, k * \delta z)$
- `Uc[i][j][k]` represents $U(t, i * \delta x, j * \delta y, k * \delta z)$

Naive Stencil Implementation and Problem Description

- Grid: $\delta x, \delta y, \delta z$ are unit distances on each direction
- `Up[i][j][k]` represents $U(t - \delta t, i * \delta x, j * \delta y, k * \delta z)$
- `Uc[i][j][k]` represents $U(t, i * \delta x, j * \delta y, k * \delta z)$
- Using Up and Uc to get $U(t + \delta t, \vec{X})$ and store it into Up

```
Up[i][j][k]  =  - Up[i][j][k]
                + c[0] * Uc[i][j][k]
                + c[1] * (Uc[i+1][j][k] + Uc[i-1][j][k])
                + c[2] * (Uc[i+2][j][k] + Uc[i-2][j][k])
                ...
```

Naive Stencil Implementation and Problem Description

- Grid: $\delta x, \delta y, \delta z$ are unit distances on each direction
- `Up[i][j][k]` represents $U(t - \delta t, i * \delta x, j * \delta y, k * \delta z)$
- `Uc[i][j][k]` represents $U(t, i * \delta x, j * \delta y, k * \delta z)$
- Using Up and Uc to get $U(t + \delta t, \vec{X})$ and store it into Up

```
Up[i][j][k]  =  - Up[i][j][k]
                + c[0] * Uc[i][j][k]
                + c[1] * (Uc[i+1][j][k] + Uc[i-1][j][k])
                + c[2] * (Uc[i+2][j][k] + Uc[i-2][j][k])
                ...
```

- `Up[i][j][k] = Stencil(Up[i][j][k], Uc)`

Naive Stencil Implementation and Problem Description

- Update rule for NAIVE stencil codes:

```
for(i = 0; i < nx; i++)
    for(j = 0; j < ny; j++)
        for(k = 0; k < nz; k++)
               Up[i][j][k] = Stencil(Up[i][j][k], Uc);
```

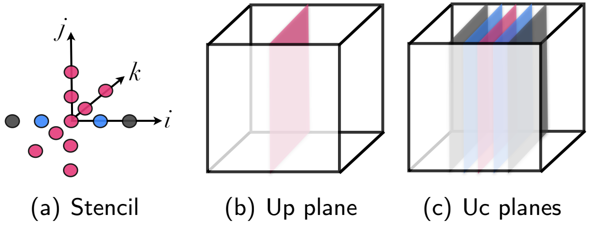


Figure: Corresponding stencil of order 2, Up[i][][] and Uc[i-2:i+2][][] planes

Naive Stencil Implementation and Problem Description

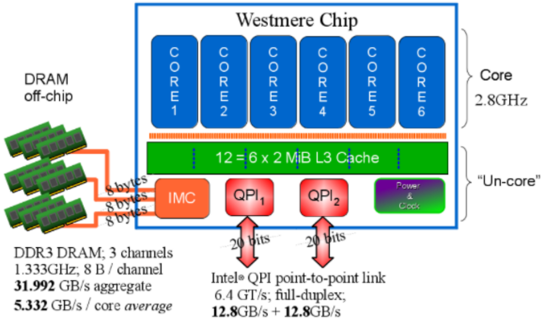


Figure: Westmere Chip[1] memory size and latency[2]: L3∼12MB(shared)/17cycles, RAM∼4GB/198cycles.

[1] http://sc.tamu.edu/systems/eos/Westmere-iDP.php

[2] Performance Tuning for CPU, Marat Dukhan

Naive Stencil Implementation and Problem Description

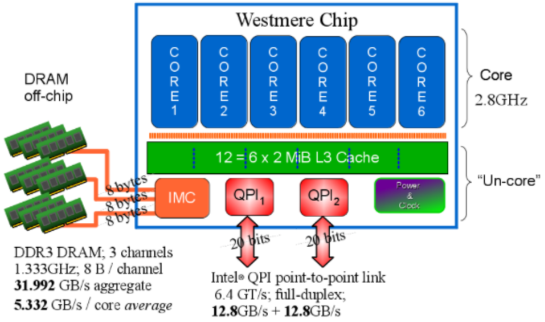


Figure: Westmere Chip[1] memory size and latency[2]: L3∼12MB(shared)/17cycles, RAM∼4GB/198cycles.

- Drawback of naive implementation
  Grid size: 260*260*260 (small problem size)
  Total memory required for storing two arrays: 134.09MB

[1] http://sc.tamu.edu/systems/eos/Westmere-iDP.php

[2] Performance Tuning for CPU, Marat Dukhan

Naive Stencil Implementation and Problem Description

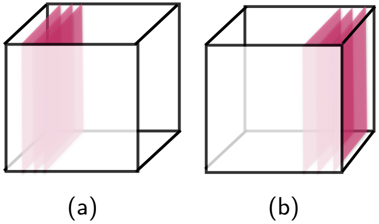


Figure: Cached data points at the start(a) of the iteration and at the end(b) of the iteration. Each array data gets evicted out of cache at each iteration.

All data array elements get loaded into the cache for at least once per iteration.

Time-Skewing Approach and Computational Analysis

- To increase data utilization in cache - cache blocking?

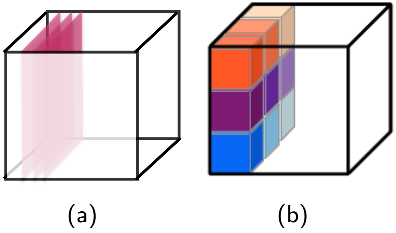


Figure: (a) Naive stencil implementation; (b) Stencil implementation by blocking in three dimensions

[3] Optimization and Performance Modeling of Stencil Computations on Modern Microprocessors. K. Datta, K. Yellick etc. *SIAM Review*, Dec. 2008

Time-Skewing Approach and Computational Analysis

- To increase data utilization in cache - cache blocking?

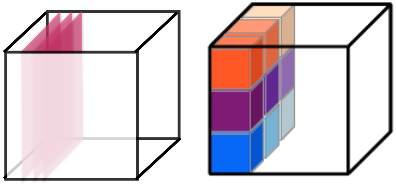


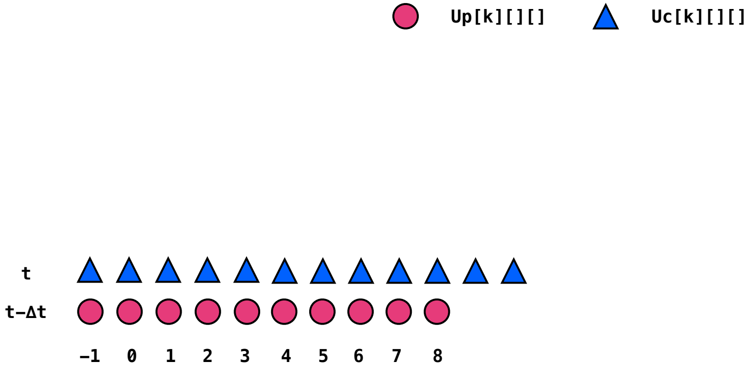
Figure: (a) Naive stencil implementation; (b) Stencil implementation by blocking in three dimensions

- But this technique is no longer effective for "practical" data sets[3]
  Because: 1. prefetching mechanism, 2. growing on-chip cache size

[3] Optimization and Performance Modeling of Stencil Computations on Modern Microprocessors. K. Datta, K. Yellick etc. *SIAM Review*, Dec. 2008

Time-Skewing Approach and Computational Analysis

- Time-Skewing Approach[4]: by blocking in time dimension, each array element gets loaded into the cache at least once every TS time steps

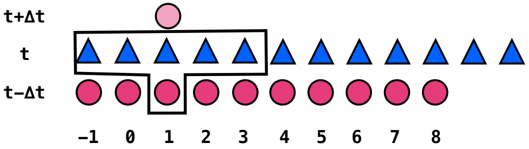


[4] David G. Wonnacott http://www.haverford.edu/computerscience/faculty//davew/

Time-Skewing Approach and Computational Analysis

- Time-Skewing Approach: by blocking in time dimension, each array element gets loaded into the cache at least once every TS time steps

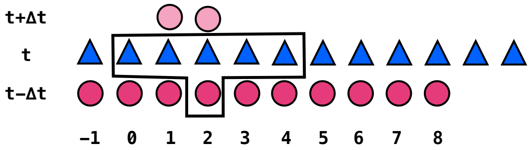
● `Up[k][][]`   ▲ `Uc[k][][]`



Time-Skewing Approach and Computational Analysis

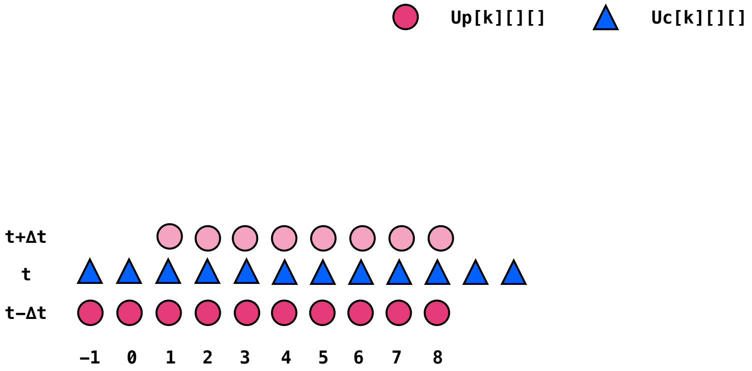
- Time-Skewing Approach: by blocking in time dimension, each array element gets loaded into the cache at least once every TS time steps

● `Up[k][][]` ▲ `Uc[k][][]`



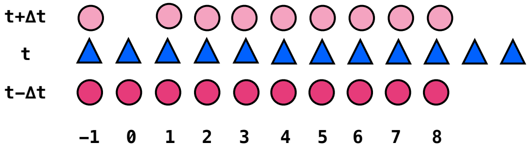
Time-Skewing Approach and Computational Analysis

- Time-Skewing Approach: by blocking in time dimension, each array element gets loaded into the cache at least once every TS time steps



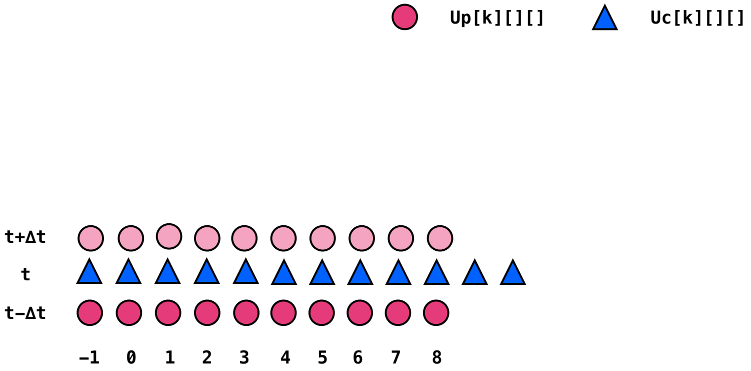
Time-Skewing Approach and Computational Analysis

- Time-Skewing Approach: by blocking in time dimension, each array element gets loaded into the cache at least once every TS time steps



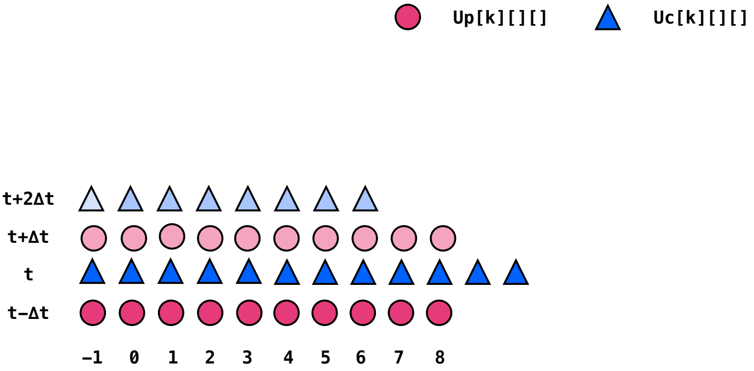
Time-Skewing Approach and Computational Analysis

- Time-Skewing Approach: by blocking in time dimension, each array element gets loaded into the cache at least once every TS time steps



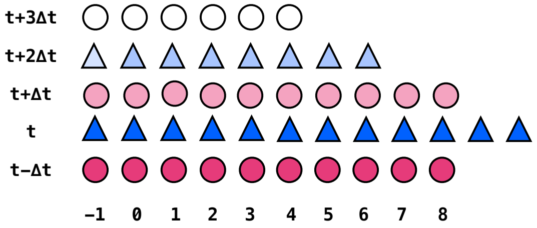
Time-Skewing Approach and Computational Analysis

- Time-Skewing Approach: by blocking in time dimension, each array element gets loaded into the cache at least once every TS time steps



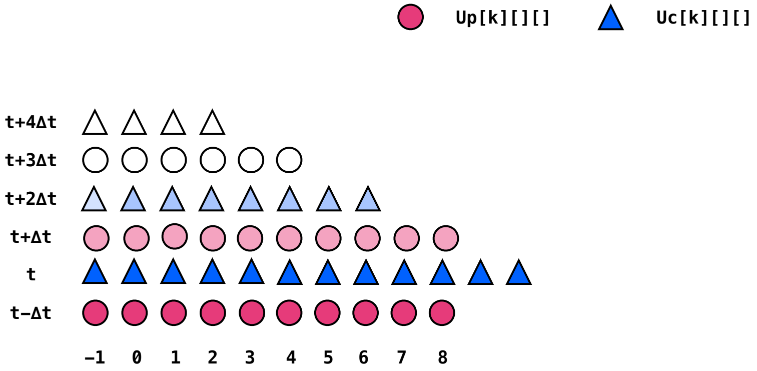
Time-Skewing Approach and Computational Analysis

- Time-Skewing Approach: by blocking in time dimension, each array element gets loaded into the cache at least once every TS time steps



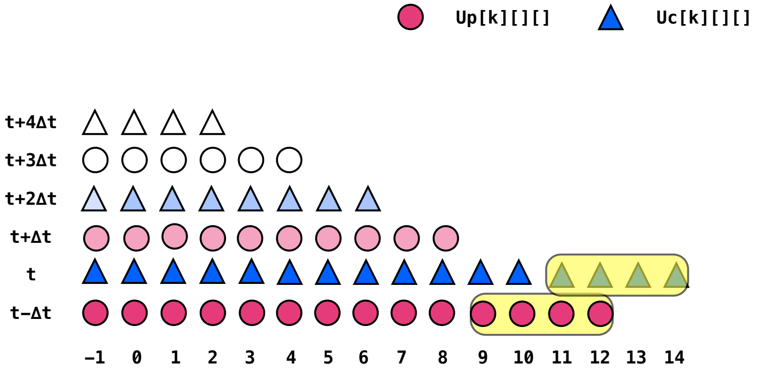
Time-Skewing Approach and Computational Analysis

- Time-Skewing Approach: by blocking in time dimension, each array element gets loaded into the cache at least once every TS time steps



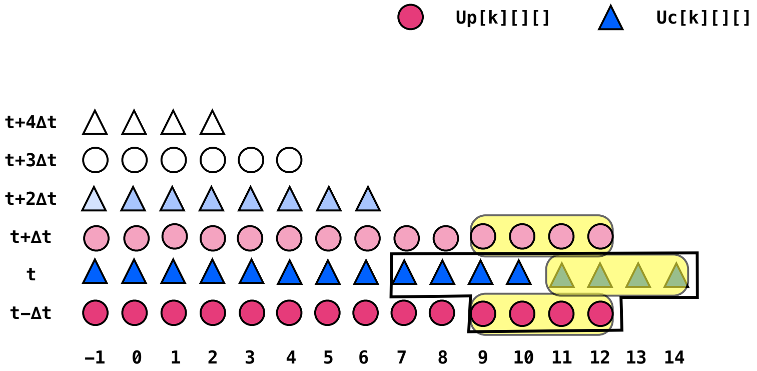
Time-Skewing Approach and Computational Analysis

- Time-Skewing Approach: by blocking in time dimension, each array element gets loaded into the cache at least once every TS time steps



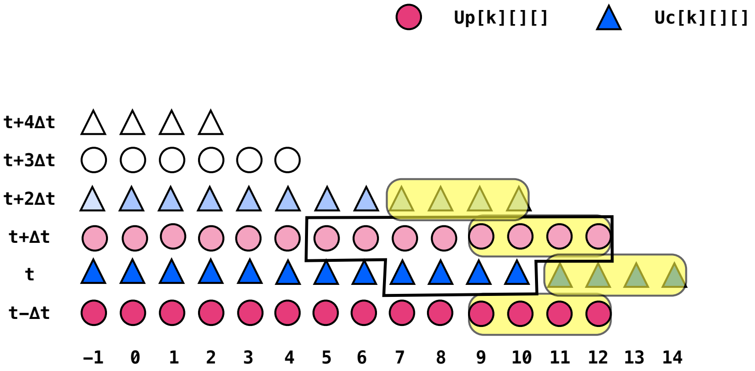
Time-Skewing Approach and Computational Analysis

- Time-Skewing Approach: by blocking in time dimension, each array element gets loaded into the cache at least once every TS time steps



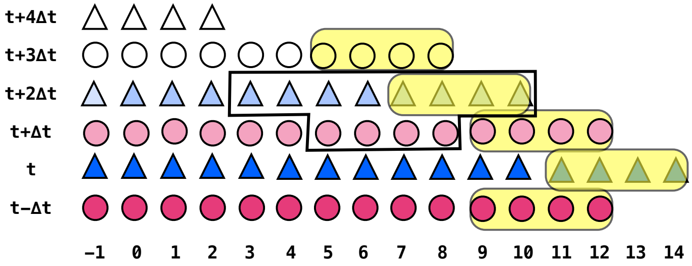
Time-Skewing Approach and Computational Analysis

- Time-Skewing Approach: by blocking in time dimension, each array element gets loaded into the cache at least once every TS time steps



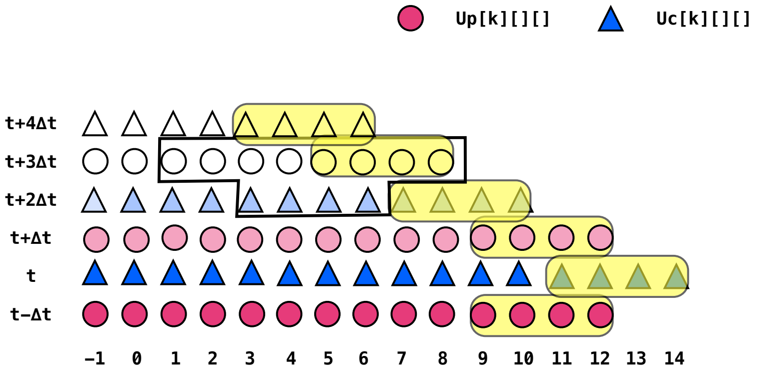
Time-Skewing Approach and Computational Analysis

- Time-Skewing Approach: by blocking in time dimension, each array element gets loaded into the cache at least once every TS time steps



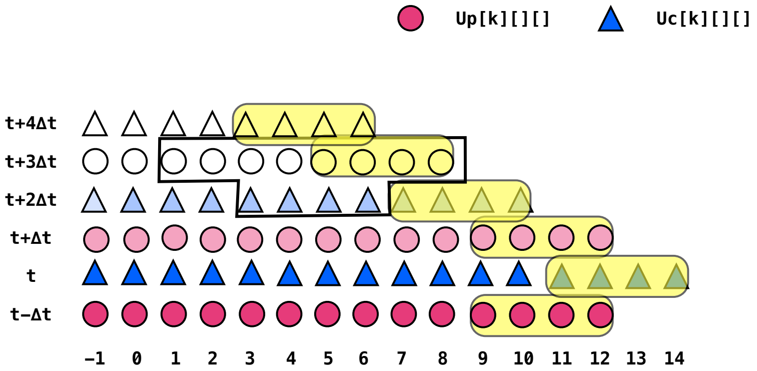
Time-Skewing Approach and Computational Analysis

- Time-Skewing Approach: by blocking in time dimension, each array element gets loaded into the cache at least once every TS time steps



Time-Skewing Approach and Computational Analysis

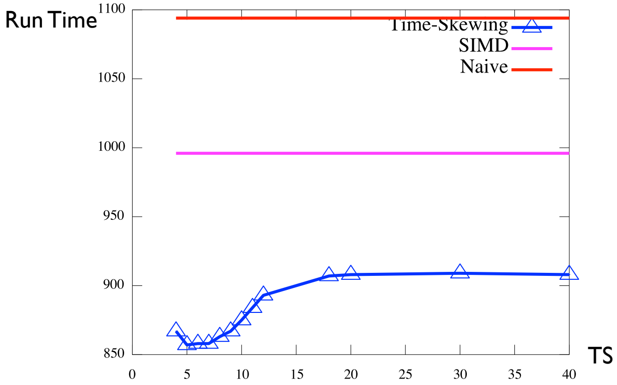
- Time-Skewing Approach: by blocking in time dimension, each array element gets loaded into the cache at least once every TS time steps



- Number of cached planes: 2WS+4TS (If WS = 4 then TS<=9)

Results

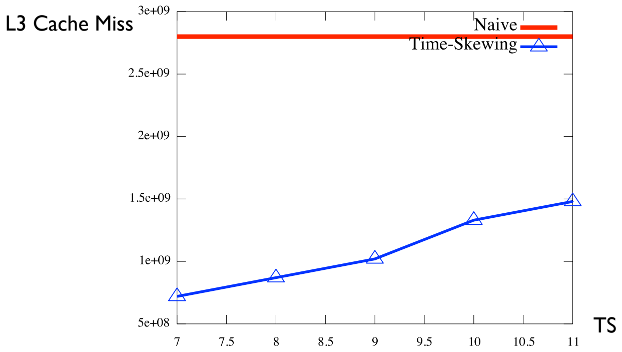
Figure: Run time results of different approaches[5].



[5] Platform: DAVinCI Westmere processor at Rice University, Compiler: icc with -O2 turned on

Results

Figure: Total number of L3 cache misses counted by PAPI via HPCTookkit[6].



[6] Platform: DAVinCI Westmere processor at Rice University, Compiler: icc with -O2 turned on, Sample period: 10000000 cycles, Overhead: <1%

Conclusion

- Apply Time-Skewing approach on ACD wave equation.
- Parameter tuning and measure the codes' performances.
- Current:
    - Integrated into IWAVE
    - Develop function to transform naive stencil codes to time-skewing code
    - Extend to higher order stencil
    - Modify the domain decomposition scheme in IWAVE
- Application: Accelerate IWAVE computation kernel

End

Thanks! Questions?