Research of improving the performance of explicit numerical methods on the x86 and ARM CPU

Authors: V. Furgailo, E. Elchinov, N. Khohlov

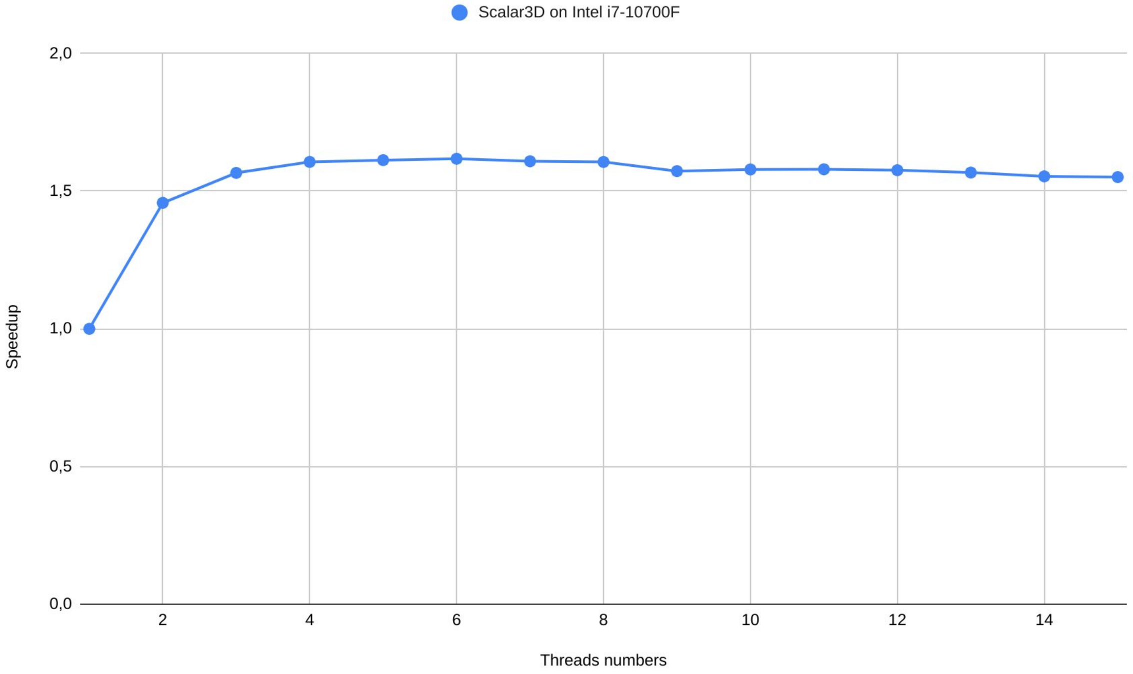
Speaker: Vladislav Furgailo

This work was performed with the financial support of the Russian Science Foundation (project No. 21-11-00139).

MIPT

Introducing

- Explicit numerical methods are used for a wide range of scientific problems
- Need to speed up stencils calculation
- Tiling for data localization
- SIMD-computing
- x86 vs. ARM



Mathematical problem

- 3D acoustic-wave propagation

$$\frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} + f(t, x, y, z)$$

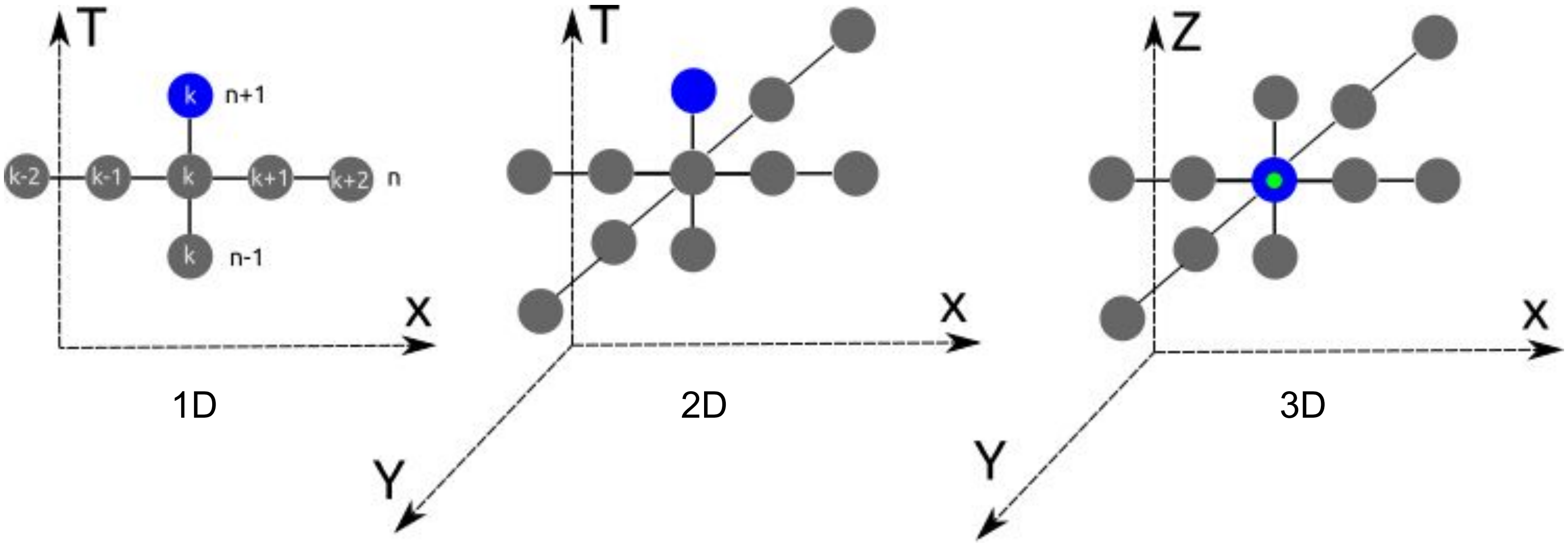
$where\ \ c - wave\ \ propagation\ \ speed,\ \ f - source$

- Finite-difference time-domain method(FDTD) was used

$$u^{n+1,k,l,m} = (3 * cc * c_0 - 2)u^{n,k,l,m} - u^{n-1,k,l,m} + cc[c_2(u^{n,k-2,l,m} + u^{n,k+2,l,m} + u^{n,k,l-2,m} + u^{n,k,l+2,m} + u^{n,k,l,m+2} + u^{n,k,l,m-2}) + c_1 * (\ldots)] + c^2 dt^2 f,$$

$where\ cc = \frac{c^2 dt^2}{dh^2};\ dt - time\ step;\ dh = dx = dy = dz - space\ step; c_0, c_1, c_2 - FD - coefficients$

Mathematical problem



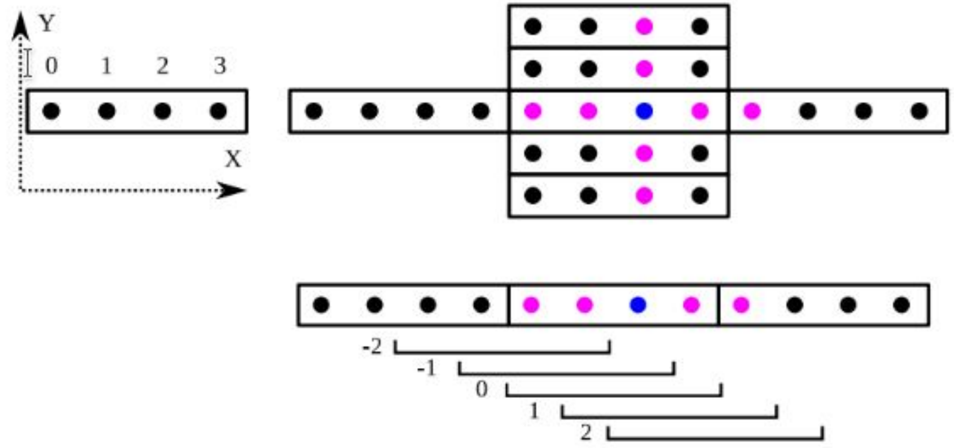
Central fourth order FD was used

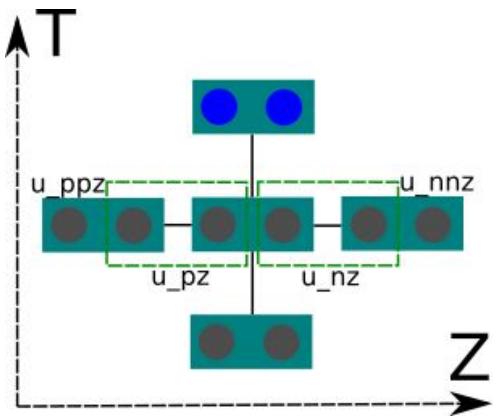
Technical detail

- Uniform 3D grid with double precision value was used
- x86: six-core Intel® Xeon® E5-2620 v2 (2.1 GHz) 32 KB (L1), 256 KB (L2), 15 MB (L3) with AVX-extension
- ARM: four-core Cortex-A53 (400 MHz) 32 KB (L1), 512 KB (L2) with NEON-extension
- C\C++: gcc 7.3.1 with OpenMP

Vectorization

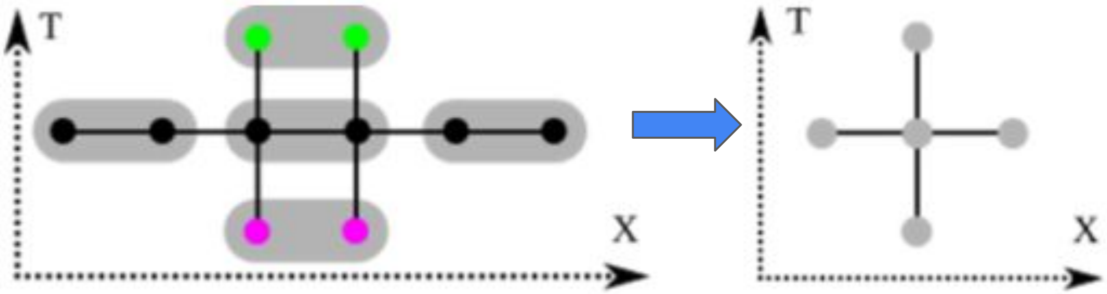
- Perform the same operation (addition or multiplication) with several data simultaneously
- AVX-instructions(_m256d register) and NEON-instructions(float64x2_t)
- Used vectorization for external spatial cycle





Order optimization

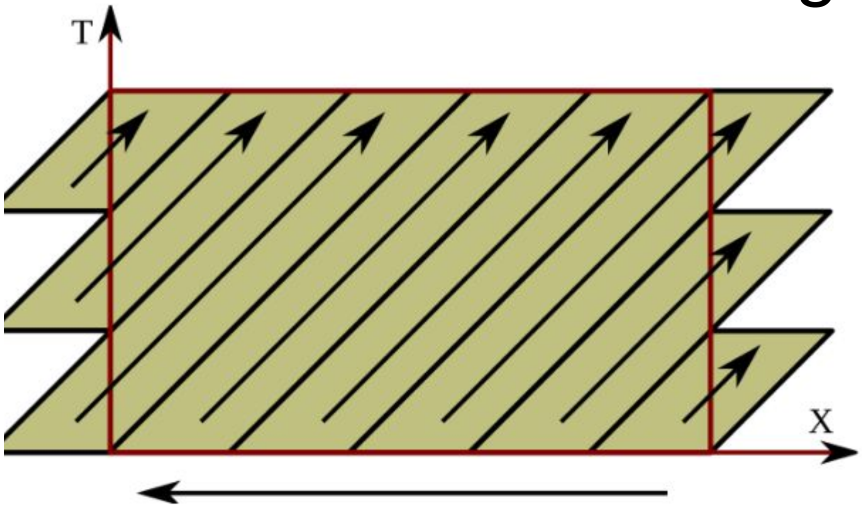
- 4th-order representation to 2th-order



Tiling

- Data localization algorithm for cache-optimization
- Non-recursive and recursive cube-tiling
- ZCube-recursive tiling

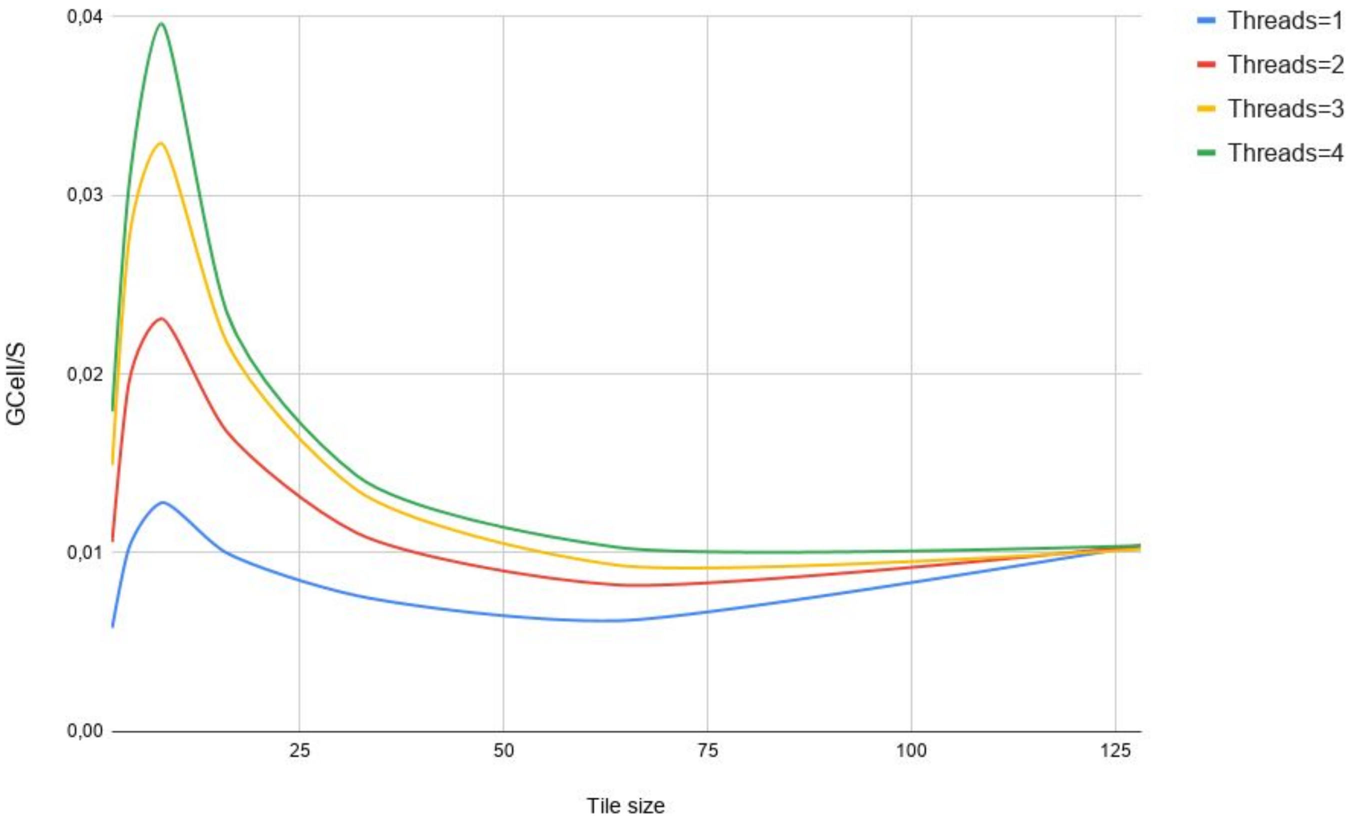
Non-recursive cube-tiling



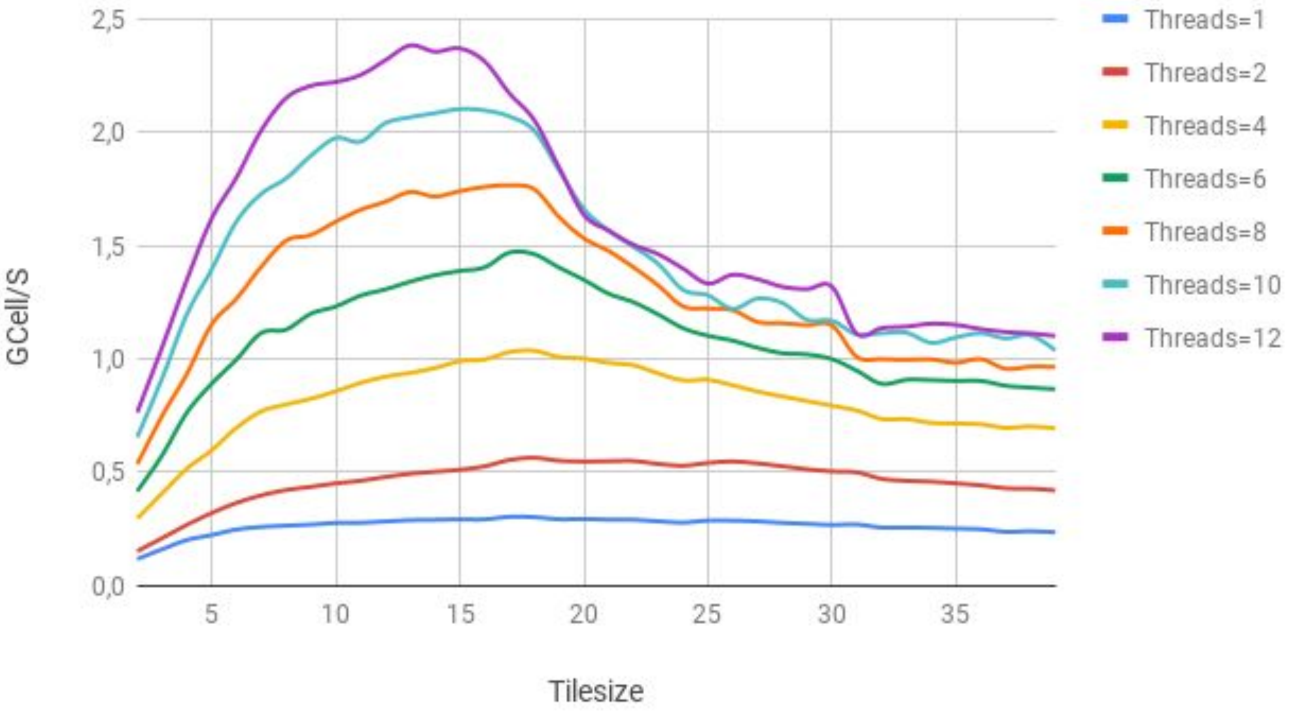

```
#define TILE_TUBE(inside_) \
    for (Int tt_ = 0; tt_ < Ts_; tt_++) {\
        for (Int tk_ = 0; tk_ < Ts_; tk_++) {\
            for (Int tj_ = 0; tj_ < Ts_; tj_++) {\
                for (Int ti_ = 0; ti_ < Ts_; ti_++) {\
                    Int t_ = t2_ * Ts_ + tt_;\
                    Int i_ = rx_ * Ts_ + ti_ + t_;\
                    Int j_ = ry_ * Ts_ + tj_ + t_;\
                    Int k_ = rz_ * Ts_ + tk_ + t_;\
                    if (inside_ || (\
                        0 <= i_ && i_ < Nx_ &&\
                        0 <= j_ && j_ < Ny_ &&\
                        0 <= k_ && k_ < Nz_ &&\
                        0 <= t_ && t_ < Nt_)) \
                    {\
                        TILED_LOOPS_BODY(i_, j_, k_, t_)\
                    }\
                }\
            }\
        }\
    }
```

Macros-substitution

Non-recursive cube-tiling. ARM

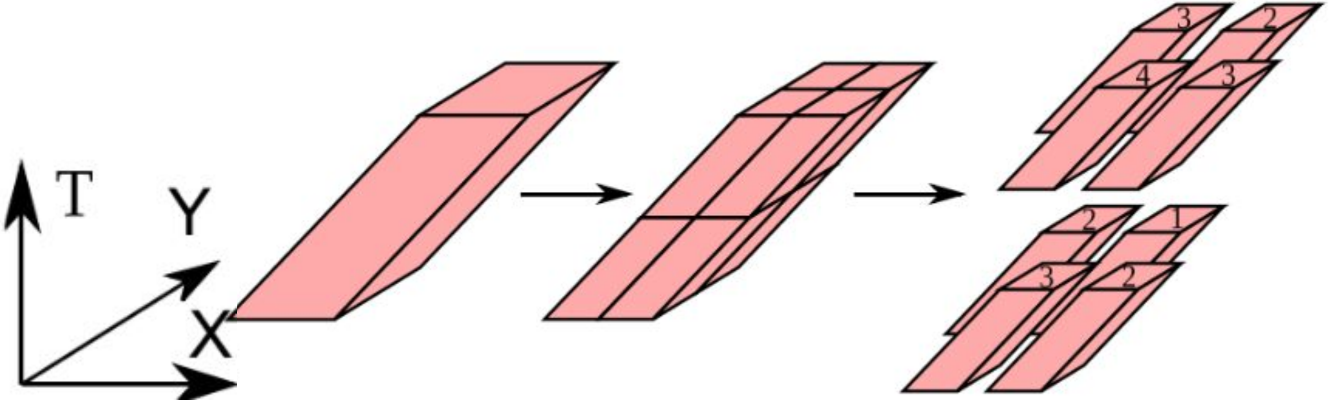


Non-recursive cube-tiling. x86



Recursive cube-tiling

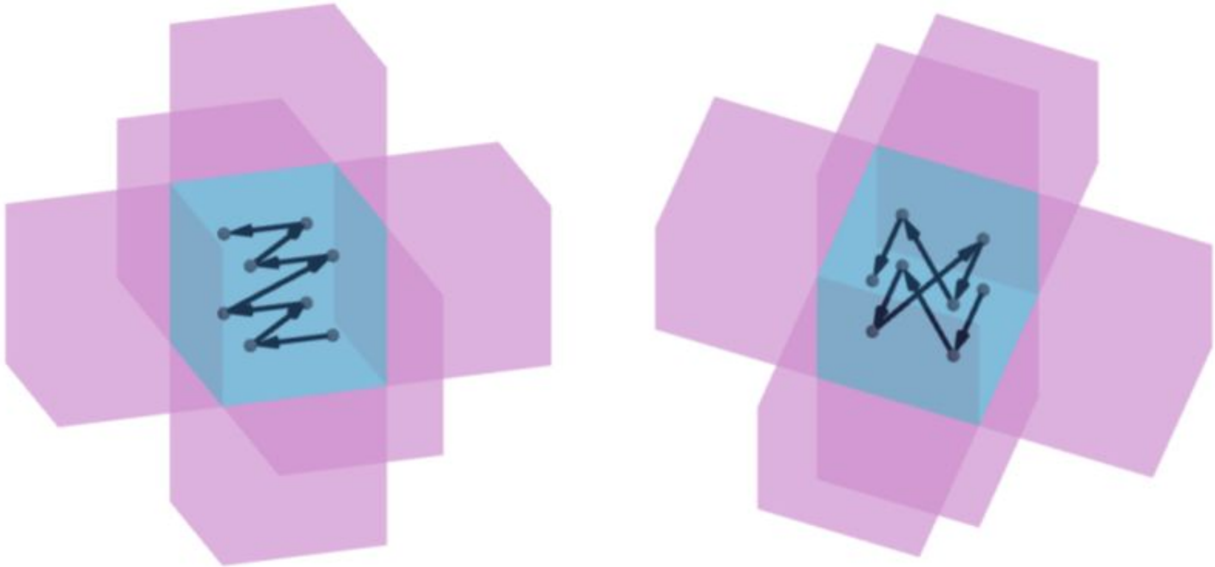
h = 2^(k−1)
the distance between tiles at the current recursion level



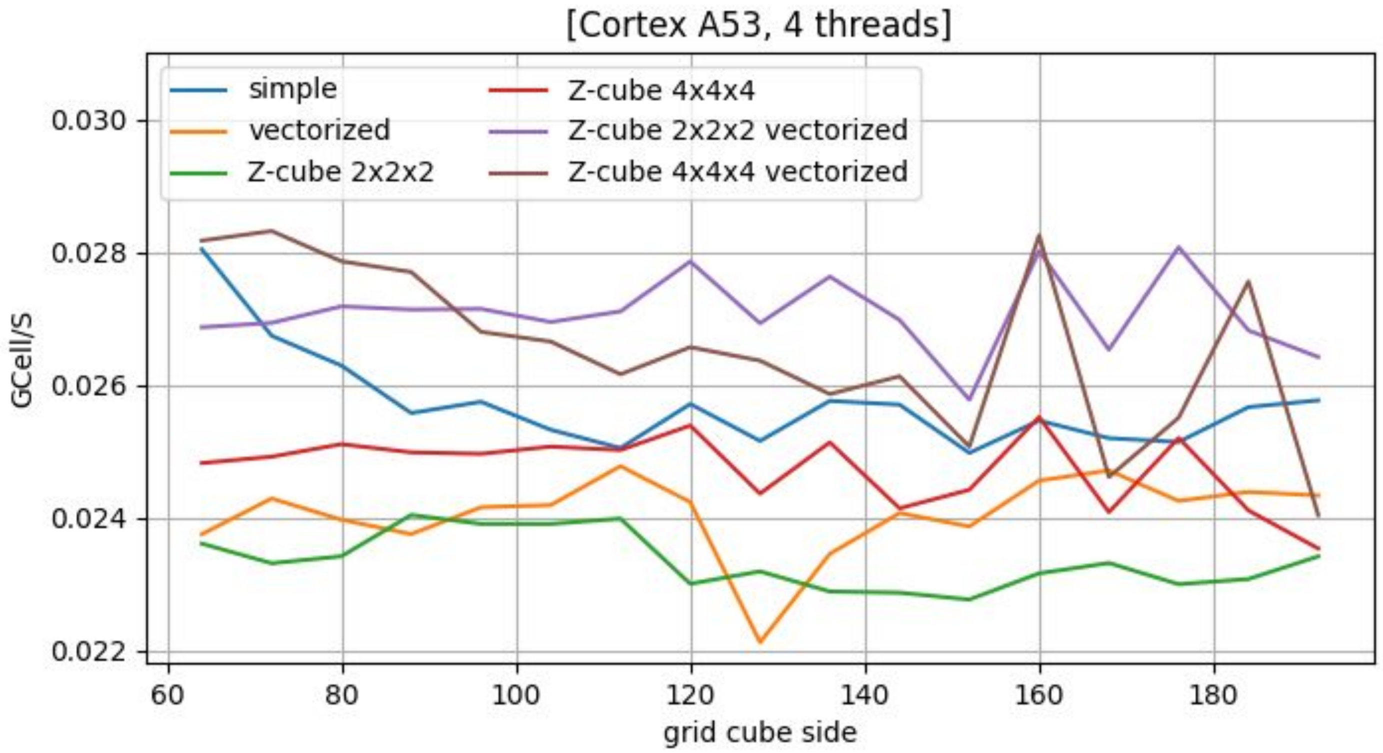

```
// generated.hpp
PROC_STENCIL_(40, 0,33, 0,41, −1,58, 0,42, 0,12, 0,44);
PROC_STENCIL_(39, 0,38, 0,46, 0,37, 0,53, 0,35, 1,3);
PROC_STENCIL_(38, −1,47, 0,39, 0,36, 0,52, 0,34, 1,2);
PROC_STENCIL_(37, 0,36, 0,44, −1,55, 0,39, 0,33, 1,1);
PROC_STENCIL_(36, −1,45, 0,37, −1,54, 0,38, 0,32, 1,0);
...
```

ZCube-tiling

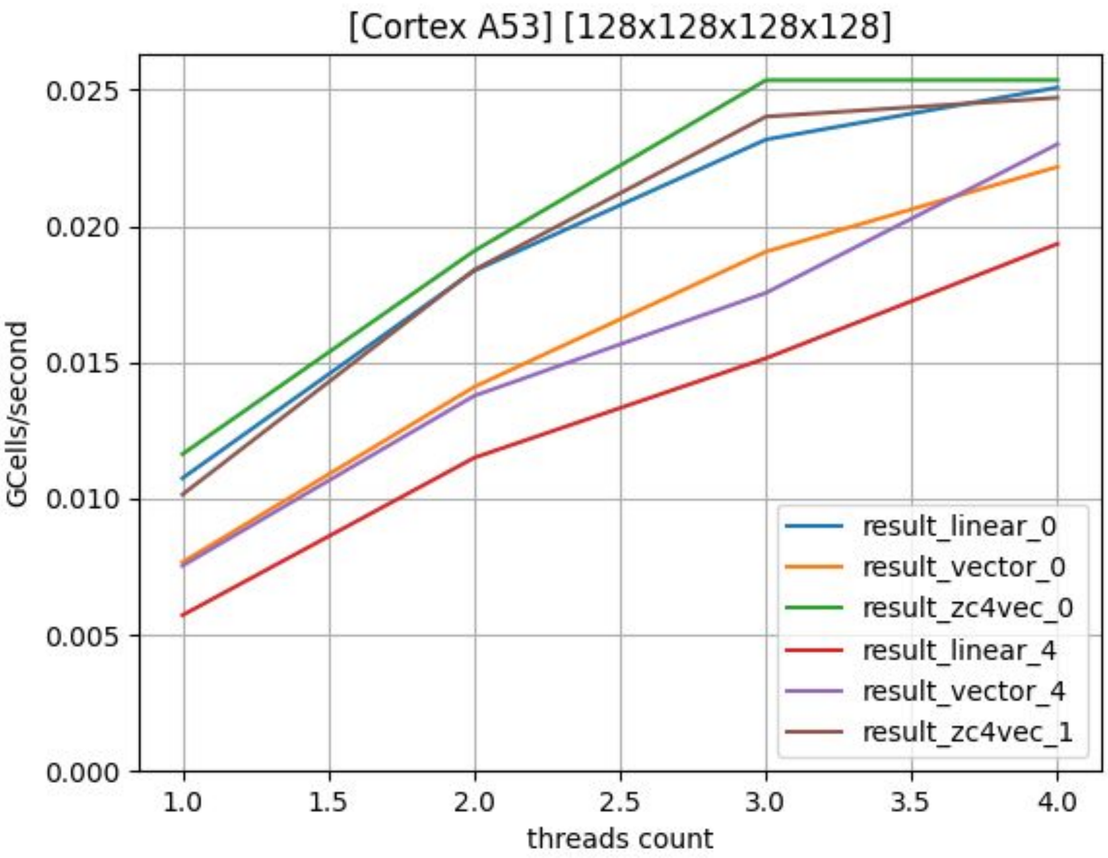
- Grid cells grouped in ZCube cells
- Vectorized ZCube + tiling



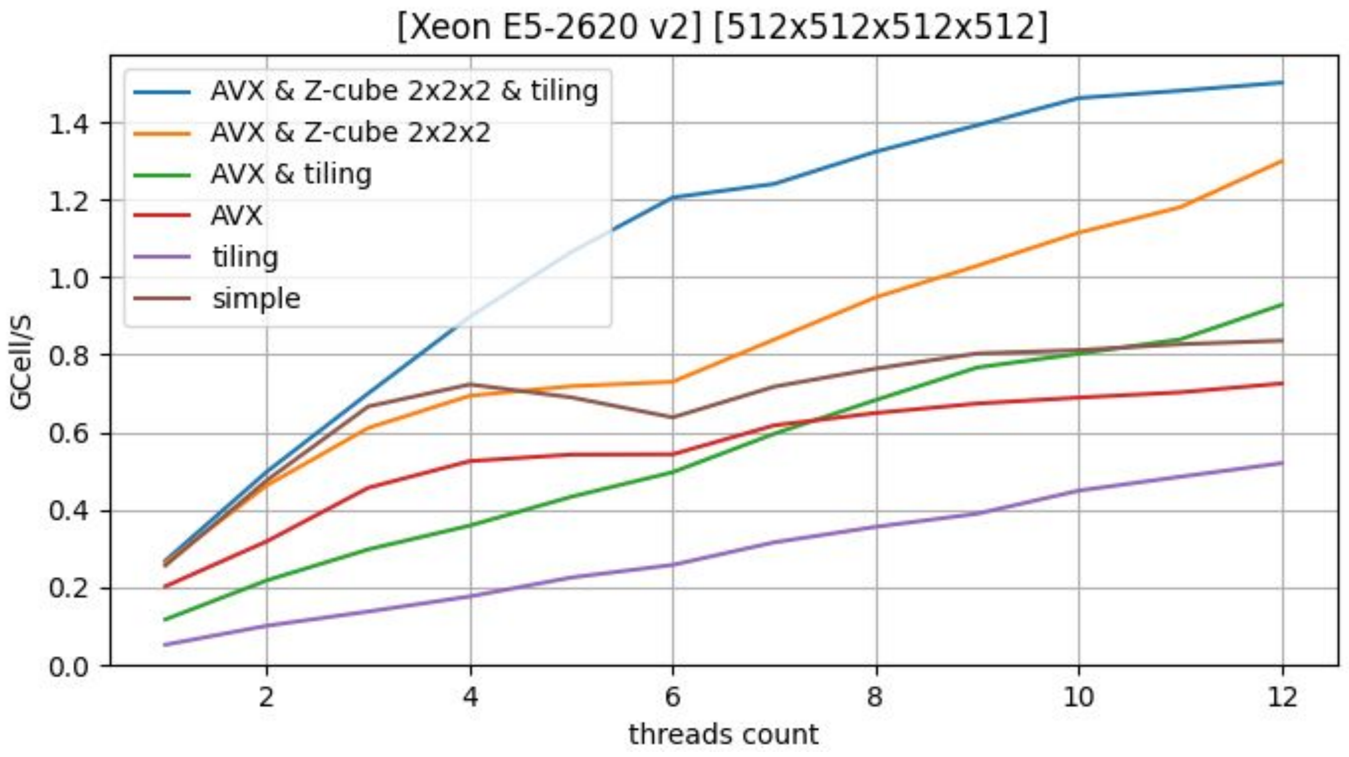
Recursive-tiling. ARM



Recursive-tiling. ARM

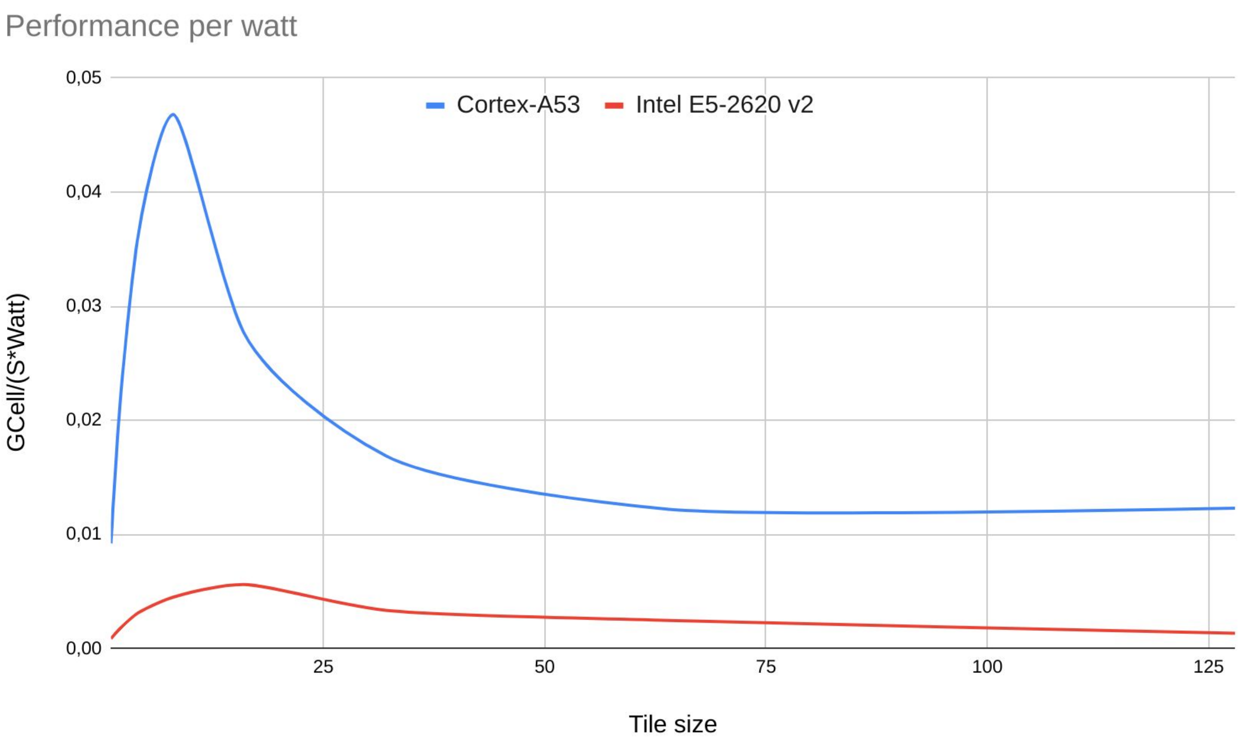


Recursive-tiling. x86



Conclusion

- Best performance with non-recursive tiling
- ARM 12 times more performance/power efficient than x86
- Cluster computing



Thank you for your attention!