

The Impact of Data Aligning on The Efficiency of a Parallel PIC Code for Numerical Simulation of Plasma Dynamics in Open Trap

Igor Chernykh, Igor Kulikov, Vitaly Vshivkov,
Ivan Chernoshtanov, Marina Boronina

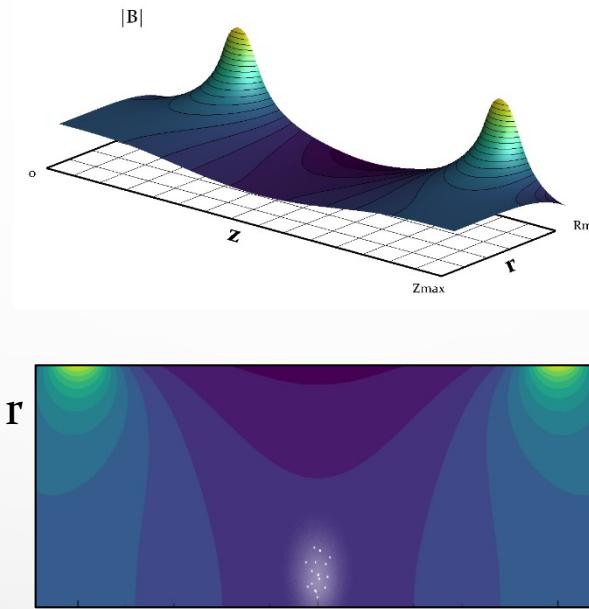
Institute of Computational Mathematics and Mathematical Geophysics SB RAS

This work was supported by RSF grant No. 19-71-20026

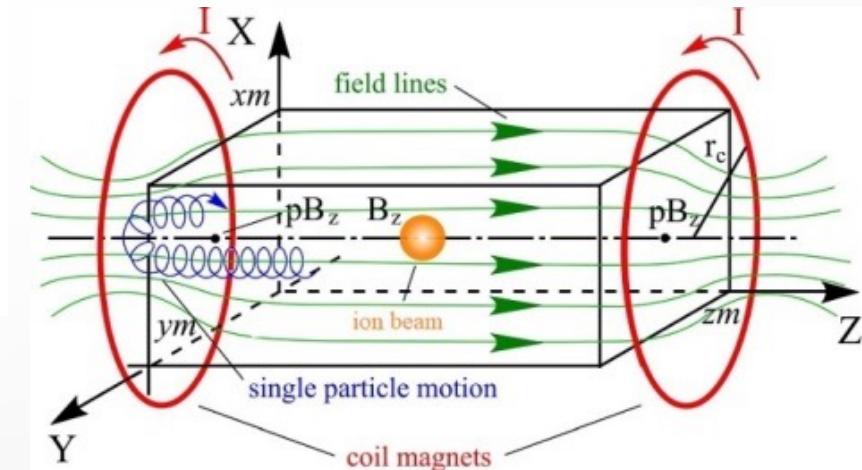
Overview of presentation

1. Overview of the problem
2. Motivation
3. Mathematical model
4. Numerical method
5. Parallel realization
6. Roofline model
7. Performance evaluation
8. Conclusion and future work

Overview of the problem



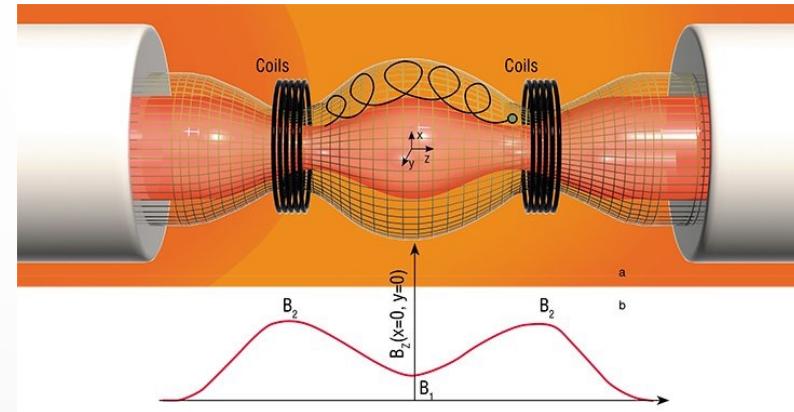
GOL3 – Plasma confinement facility with the external magnetic field generation



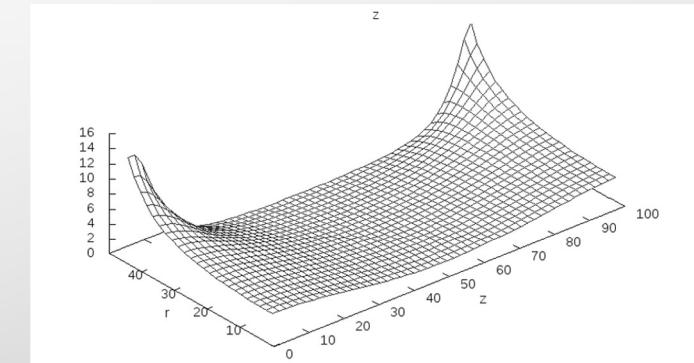
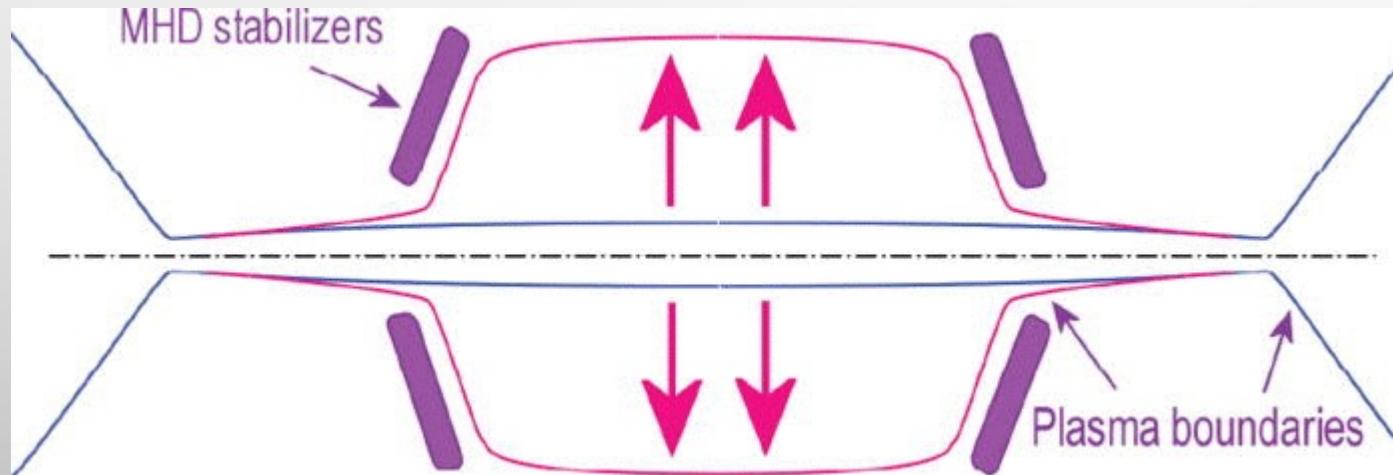
1. Cylindrical volume filled by ions
2. Two magnetic field sources
3. Injection of electrons at the center

https://en.wikipedia.org/wiki/Budker_Institute_of_Nuclear_Physics

Overview of the problem



A. Shosin 23 Apr 2018 , Budker's Universe , volume Special Issue, N1



beam injection

A. D. Beklemishev **Diamagnetic “bubble” equilibria in linear traps** Phys. Plasmas, 23, 082506 (2016)

Motivation

Simulate all parameters (magnetic field, power of electrons injection, etc) to find stable mode of GOL plasma confinement facility.



Diamagnetic equilibrium for thermonuclear fusion

Mathematical model

Vlasov kinetics equation (ions)

$$\frac{\partial f_i}{\partial t} + \vec{v} \frac{\partial f_i}{\partial \vec{r}} + \frac{\vec{F}_i}{m_i} \frac{\partial f_i}{\partial \vec{v}} = 0$$

$$\vec{F}_i = e \left(\vec{E} + \frac{1}{c} [\vec{v}, \vec{B}] \right) + R_i$$

$$n_i(\vec{r}) = \int f_i(t, \vec{r}, \vec{v}) d\vec{v}$$

$$\vec{V}_i(\vec{r}) = \frac{1}{n_i(\vec{r})} \int \vec{v} f_i(t, \vec{r}, \vec{v}) d\vec{v}$$

MHD equations (electrons)

$$-e\vec{E} - \frac{e}{c} [\vec{V}_e, \vec{B}] - \frac{\nabla p_e}{n_e} + \vec{R}_e = m_e \frac{d\vec{V}}{dt}$$

$$\vec{R}_e = -m_e \frac{\vec{V}_e - \vec{V}_i}{\tau_{ei}}$$

Maxwell equations

$$\frac{1}{c} \frac{\partial \vec{E}}{\partial t} = \text{rot} \vec{B} - \frac{4\pi}{c} \vec{j}$$

$$\frac{1}{c} \frac{\partial \vec{B}}{\partial t} = -\text{rot} \vec{E}$$

$$\text{div} \vec{E} = 4\pi\rho$$

$$\text{div} \vec{B} = 0$$

Heat equations

$$n_e \left(\frac{\partial T_e}{\partial t} + (\vec{V}_e \nabla) T_e \right) = (\gamma - 1) (Q_e - \text{div} \vec{q}_e - p_e \text{div} \vec{V}_e)$$

$$Q_e = \frac{j^2}{\sigma}$$

State equation

$$p_e = n_e T_e$$

$$n_i = n_e$$

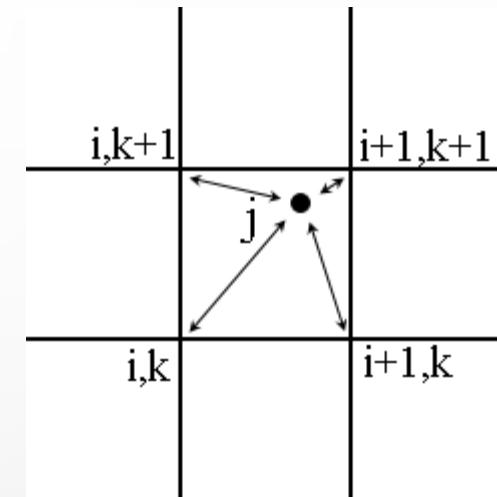
$$R_i = -R_e$$

$$\vec{j} = e(n_i \vec{V}_i - n_e \vec{V}_e)$$

$$\rho = e(n_i - n_e)$$

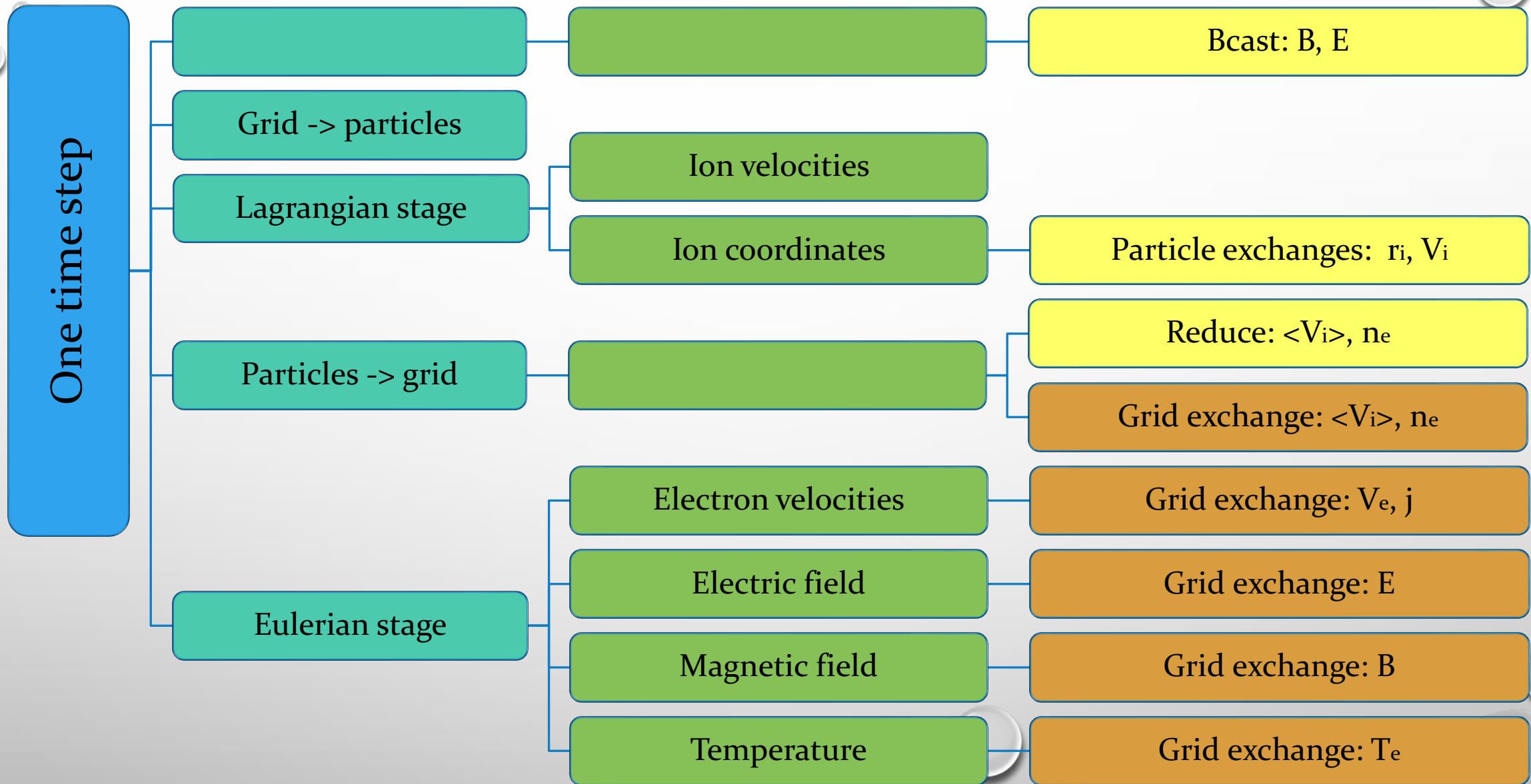
Numerical method (Finite-Difference Time-Domain (FDTD))

- ✓ Cylinder coordinate system due to the axial symmetry, $r=0$ is singularity
- ✓ Hybrid model, the ions are described kinetically, the electrons by MHD
- ✓ Particles-in-cell method with PIC form-factor
- ✓ Grids shifted by half a step
- ✓ Mixed Eulerian-Lagrangian decomposition for the parallelization

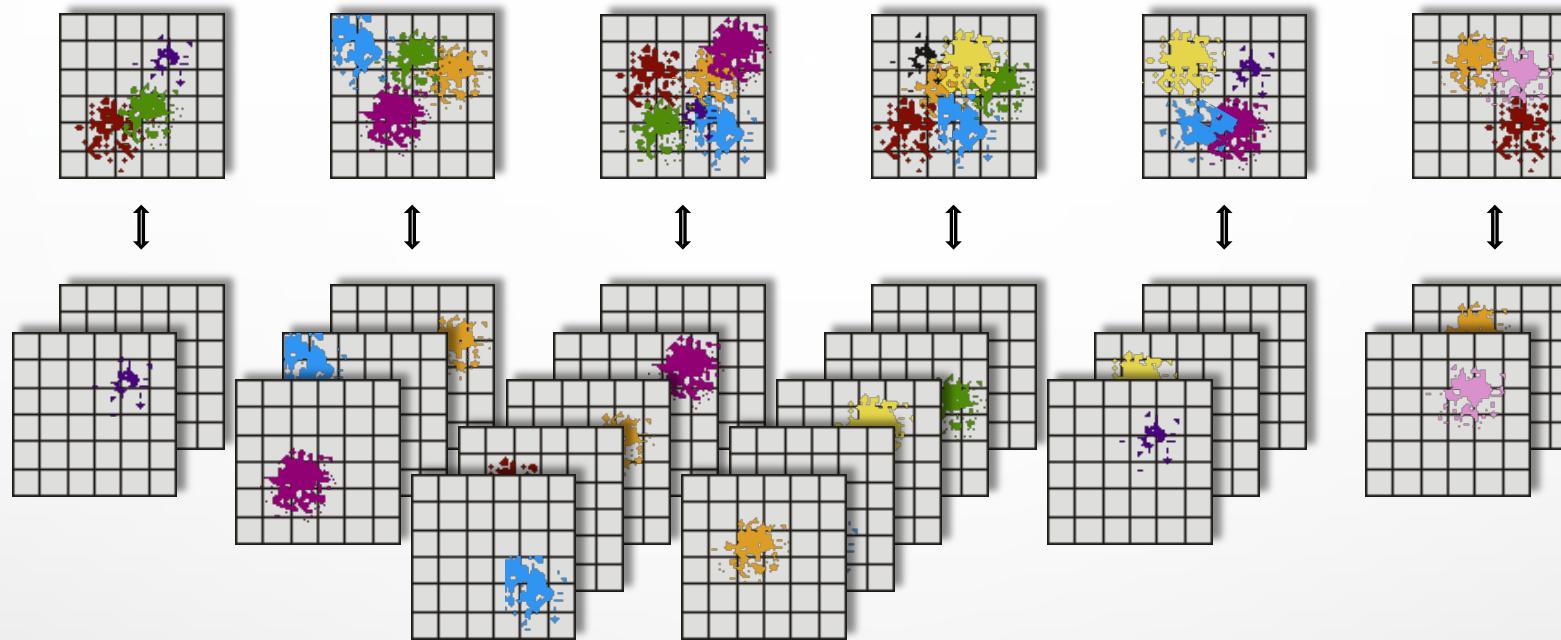


particles >> cells

Parallel realization



Parallel realization



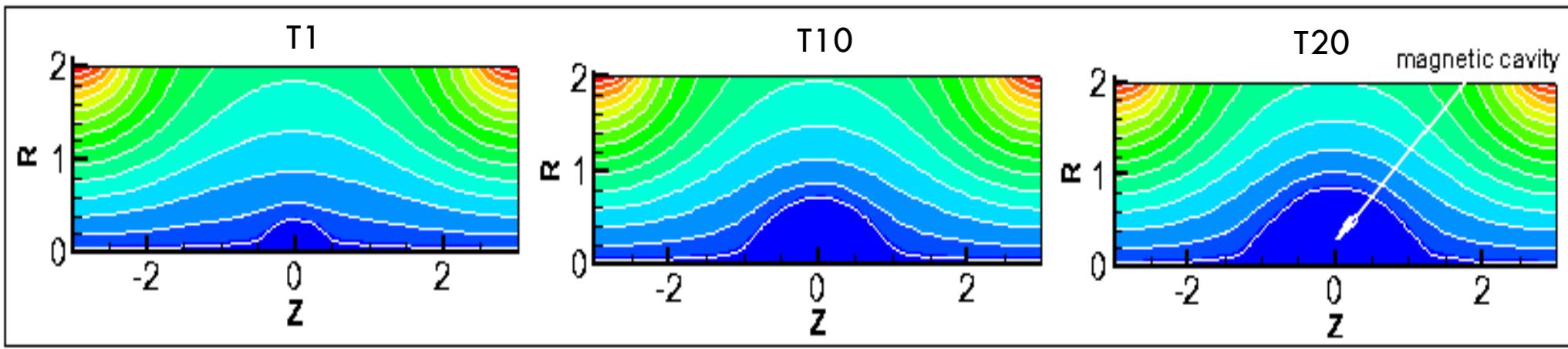
At the initial moment, the background particles are distributed within the group.

The injected particle is written to the next nucleus in its group.

When leaving the subarea, the array of particles is sent to one of the processors of the neighboring group.

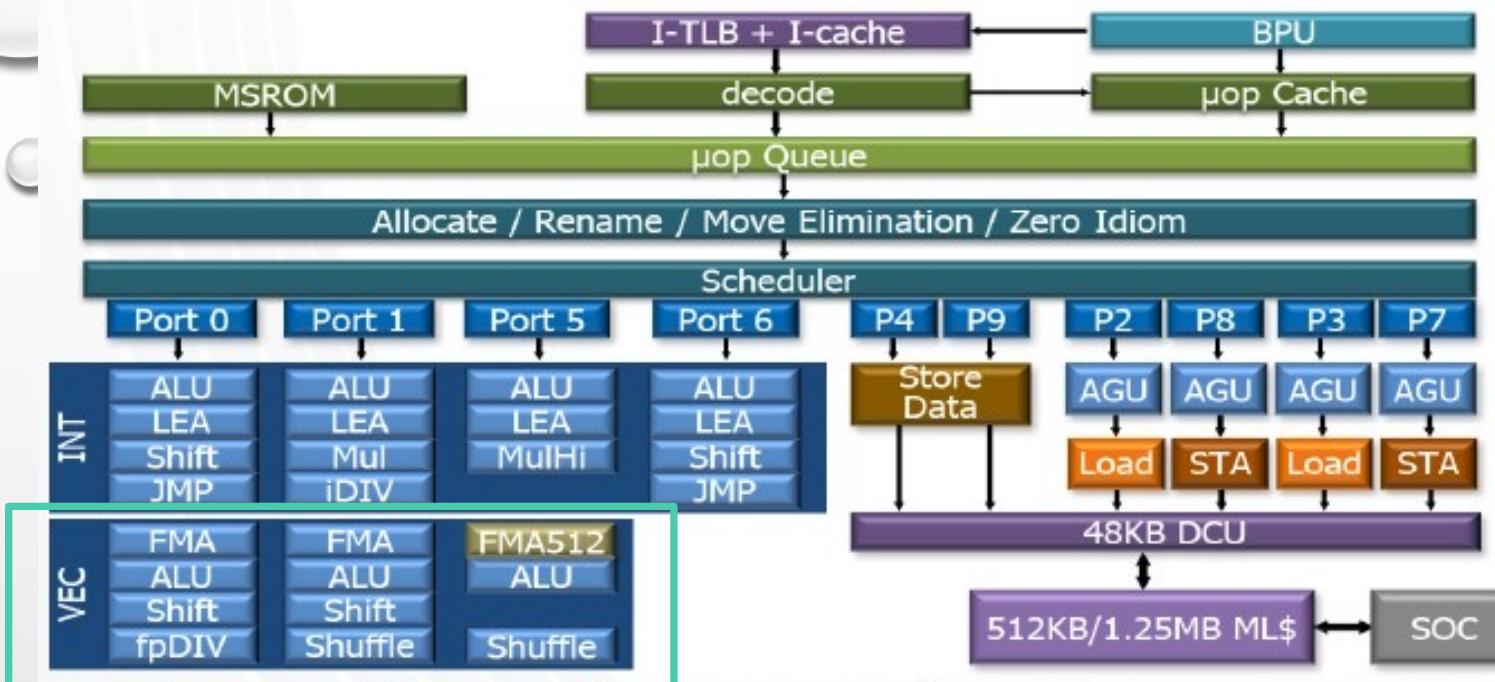
Parallel realization

B



It is shown, that the self-consistent interaction of the beam with the plasma leads to the formation of a magnetic cavity, the size of which depends on the characteristics of the ion beam and determines the volume and density of the accumulated plasma.

G.I. Dudnikova, M.A. Boronina, E.A. Genrikh, V.A. Vshivkov. Computer simulation of ion beam-plasma interaction // Journal of Physics: Conference Series -2019.-V. 1393. – 012041.

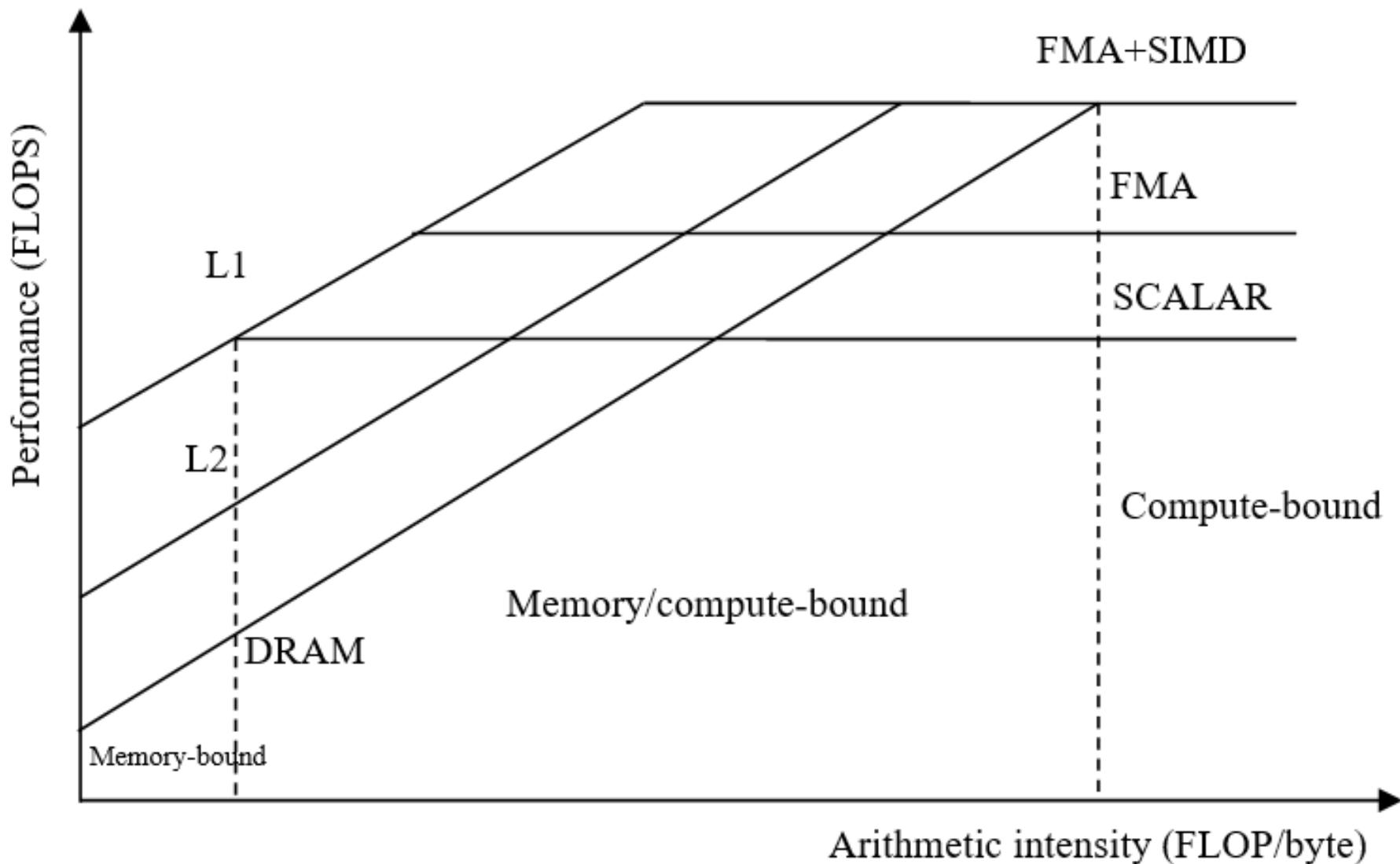


	Cascade Lake (per core)	Ice Lake (per core)
Out-of-order Window	224	384
In-flight Loads + Stores	72 + 56	128 + 72
Scheduler Entries	97	160
Register Files – Integer + FP	180 + 168	280 + 224
Allocation Queue	64/thread	70/thread
L1D Cache (KB)	32	48
L1D BW (B/Cyc) – Load + Store	128 + 64	128 + 64
L2 Unified TLB	1.5K	2K
Mid-level Cache (MB)	1	1.25

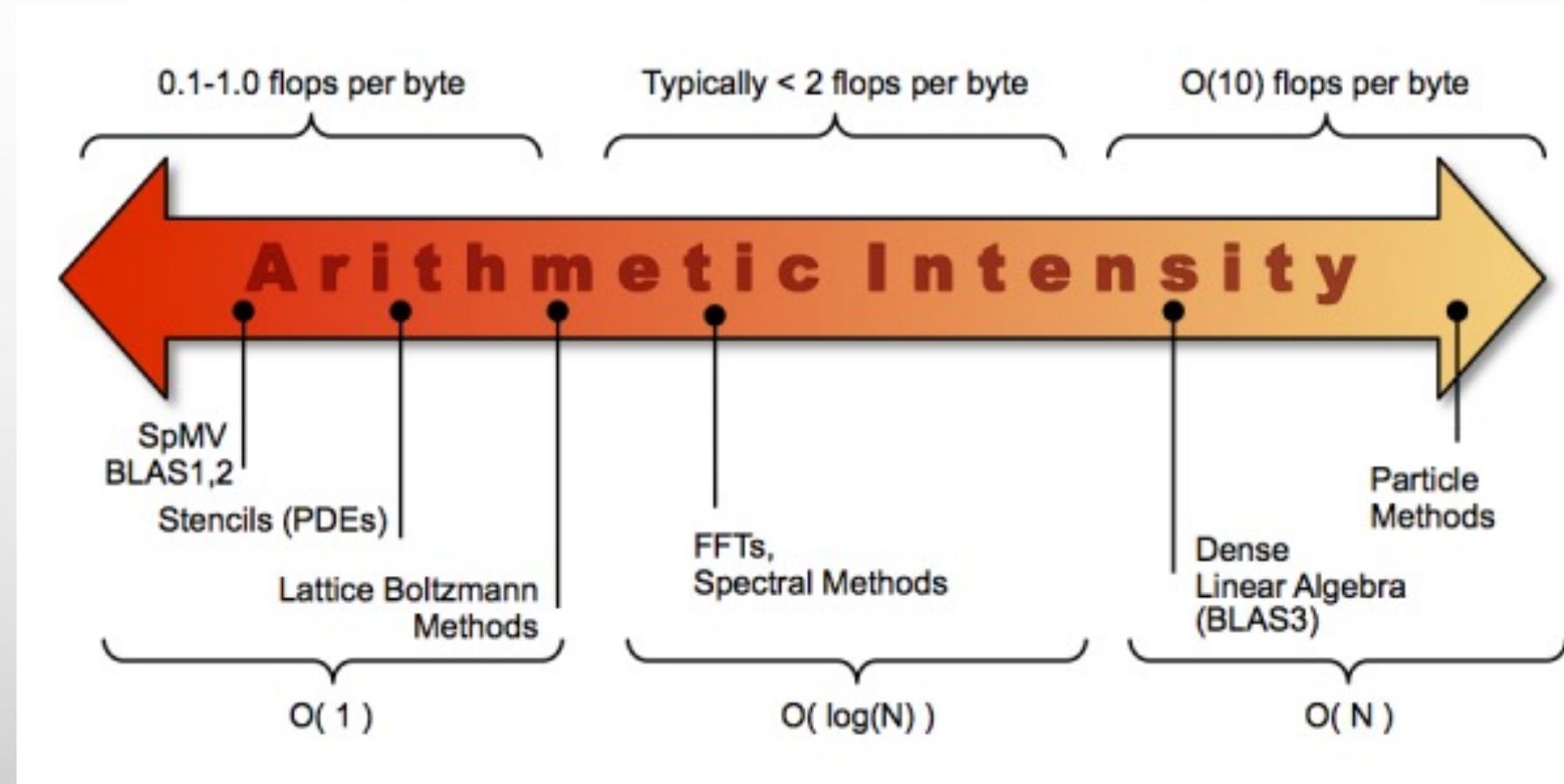
- Improved Front-end: higher capacity and improved branch predictor
- Wider and deeper machine: wider allocation and execution resources + larger structures
- Enhancements in TLBs, single thread execution, prefetching
- Server enhancements – larger Mid-level Cache (L2) + second FMA

~18% Increase In IPC On Existing SPECcpu2017(est) Integer Rate Binaries

Roofline model



Roofline model



Baghudana, Ashish & Kesiz Abnousi, Vartan. (2018). A Parallel Algorithm for LDA using Stochastic Collapsed Variational Bayesian Inference.

Siberian Supercomputer Center

NKS-1P (RSC, hot water cooling, 2448 cores, ~182TFLOPS Rpeak):

- 27 nodes: 2 CPU Intel Xeon E5-2697v4 [128 GB DDR4, 256 GB DDR4] (864 cores, 2.6GHz) (1 узел 2x375GB Intel Optane [IMDT])
- 16 nodes: 1 CPU Intel Xeon Phi 7290 KNL [16 GB MCDRAM+96 GB DDR4] (1152 cores, 1.5-1.7 GHz)
- 1 node: 2 CPU Intel Xeon Platinum 8268 [192 GB DDR4] (48 cores, 2.9 GHz)
- 8 nodes: 2 CPU Intel Xeon Gold 6248R [192/384/768 GB DDR4] (384 cores, 2.9 GHz)
- Intel OmniPath 100 Gb/s
- Intel Lustre – 200 TB + NFS 100TB

NKS-30T (HP, air cooling, ~1500 CPU cores (2.9GHz), ~30000 GPU cores, 85TFLOPS (GPU) + 22TFLOPS (CPU)):

- 576 CPU Intel Xeon E5450/E5540(2688 cores)
- 80 CPU Intel Xeon X5670(480 cores)
- 120 GPU NVIDIA Tesla M 2090(61440 cores)
- Infiniband QDR 40 Gb/s
- HP Ibrix – 90 TB



Performance evaluation

Problems with code: else if statements, difficult data balancing, arrays

Structure of arrays and data alignment:

```
type coo
real*8, allocatable :: r(:)
real*8, allocatable :: z(:)
real*8, allocatable :: u(:)
real*8, allocatable :: v(:)
real*8, allocatable :: w(:)
real*8, allocatable :: a(:)
real*8, allocatable :: it(:)
dir$ attributes align:64 :: r
dir$ attributes align:64 :: z
dir$ attributes align:64 :: u
dir$ attributes align:64 :: v
dir$ attributes align:64 :: w
dir$ attributes align:64 :: a
end type coo
```

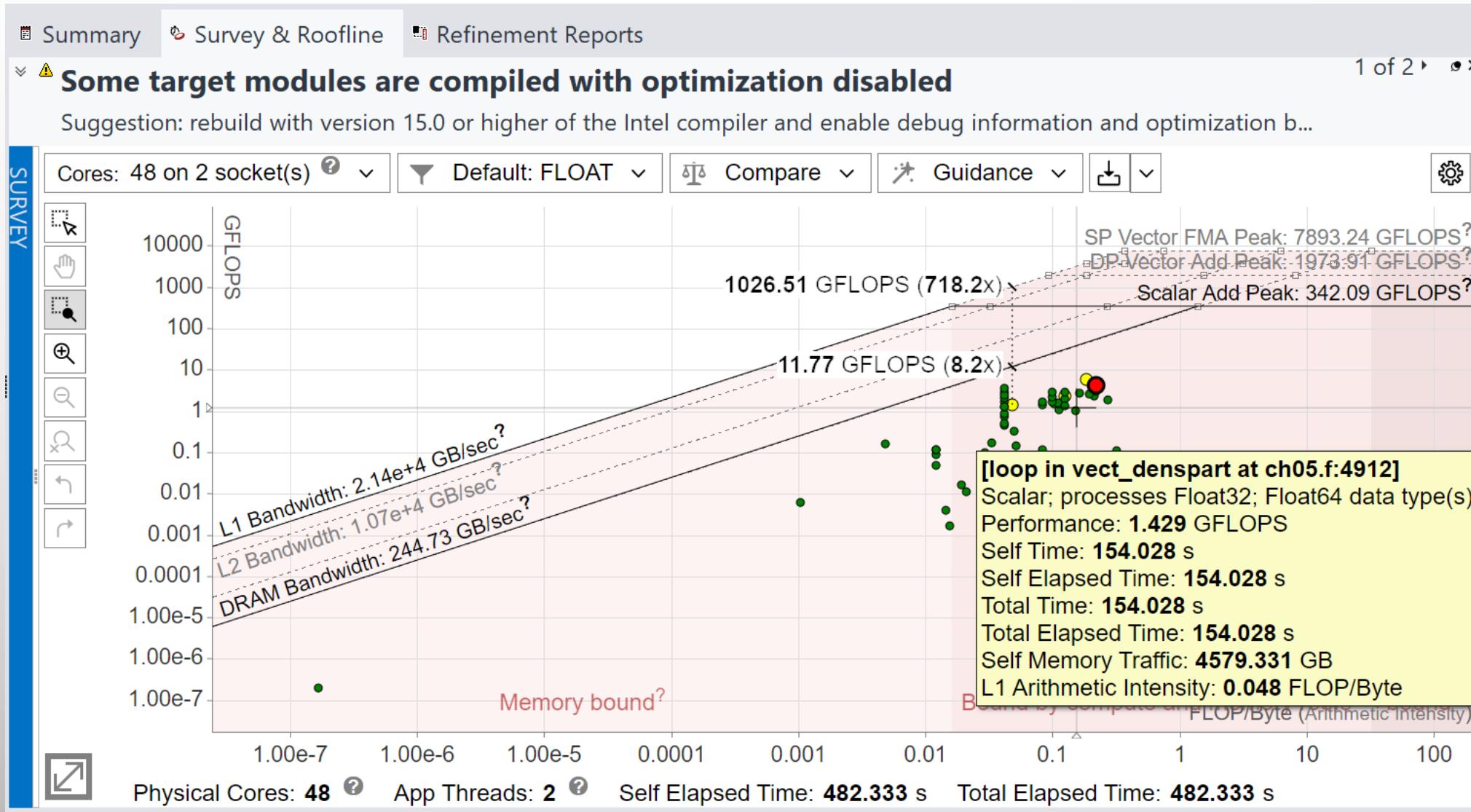
Three versions of code:

1. No aligned data , no simd
instructions, Intel Fortran
compiler 2019 mpiifort

2. Aligned data, AVX512, Intel
Fortran compiler 2019
mpiifort -xcascadelake

3. No aligned data, AVX512, Intel
Fortran oneAPI 2021 compiler
mpiifort -xcascadelake

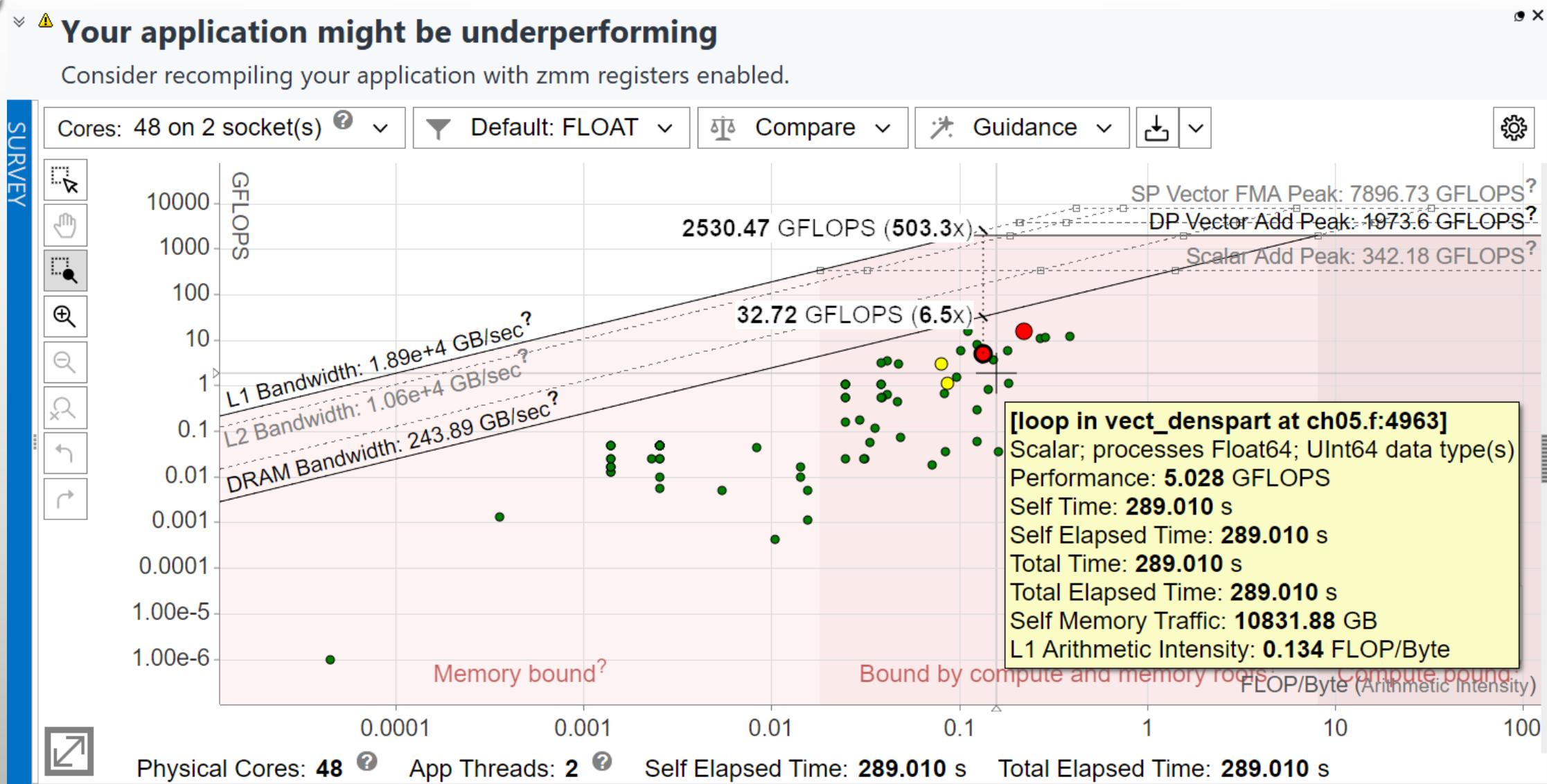
Roofline model of parallel code (2x6248R, nonaligned, nonsimd, Intel Fortran compiler 2019 (1.23 GFLOPS)



Roofline model of parallel code (2x6248R, aligned, AVX-512, Intel Fortran compiler 2019) 1.86 GFLOPS

⚠ Your application might be underperforming

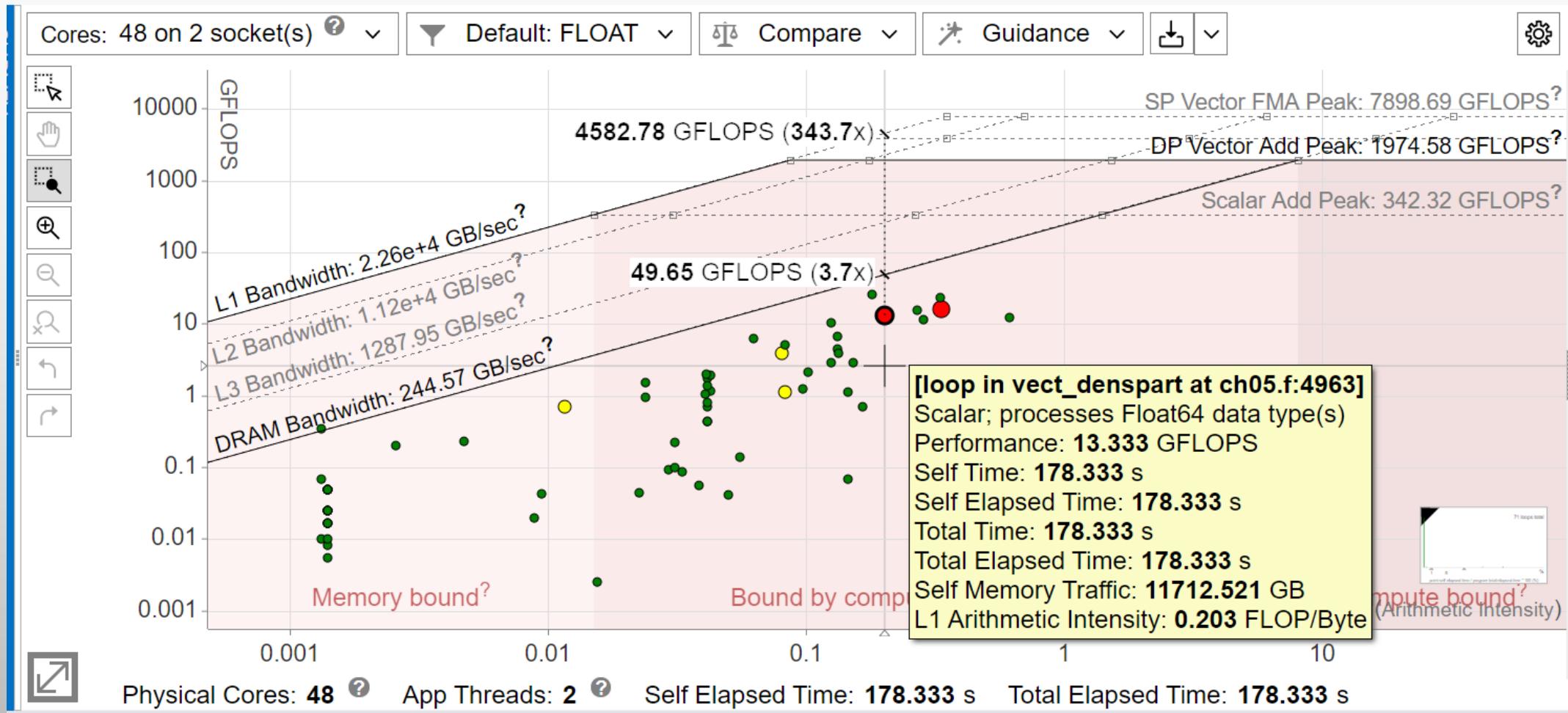
Consider recompiling your application with zmm registers enabled.



Roofline model of parallel code (2x6248R, nonaligned, AVX-512, Intel Fortran oneAPI 2021) 2.66 GFLOPS

⚠ Your application might be underperforming

Consider recompiling your application with zmm registers enabled.



Performance evaluation

Particles density calculation function (Eulerian stage):

1. **vec_denspart (nonaligned, nonsimd, Intel Fortran compiler 2019)** – 1.43 GFLOPS
2. **vec_denspart(aligned, AVX512, Intel Fortran compiler 2019)** – 5 GFLOPS **(3.5x)**
3. **vec_denspart (nonaligned, AVX512, Intel Fortran oneAPI 2021)** – 13.33 GFLOPS **(2.66x)**

Structure of arrays, data alignment, compiler autovectorization

Total code performance:

1. **PIC code (nonaligned, nonsimd, Intel Fortran compiler 2019)** – 1.23 GFLOPS
2. **PIC (aligned, AVX512, Intel Fortran compiler 2019)** – 1.86 GFLOPS **(+51%)**
3. **PIC (nonaligned, AVX512, Intel Fortran oneAPI 2021)** – 2.66 GFLOPS **(+43%)**

Conclusion

1. Update your system software and compilers.
2. Use advanced vector instructions (AVX2, AVX512, AMX) together with MPI and OpenMP in your code.
3. Use all compiler optimization options to build the fastest code for your CPU architecture. (`mpiifort -xCOMMON-AVX512`)
4. Use FMA commands

COMMON-AVX512

May generate Intel® Advanced Vector Extensions 512 (Intel® AVX-512) Foundation instructions, Intel® AVX-512 Conflict Detection Instructions (CDI), as well as the instructions enabled with CORE-AVX2. Optimizes for Intel® processors that support Intel® AVX-512 instructions.

CORE-AVX512

May generate Intel® Advanced Vector Extensions 512 (Intel® AVX-512) Foundation instructions, Intel® AVX-512 Conflict Detection Instructions (CDI), Intel® AVX-512 Doubleword and Quadword Instructions (DQI), Intel® AVX-512 Byte and Word Instructions (BWI) and Intel® AVX-512 Vector Length Extensions (VLE), as well as the instructions enabled with CORE-AVX2. Optimizes for Intel® processors that support Intel® AVX-512 instructions.

CORE-AVX2

May generate Intel® Advanced Vector Extensions 2 (Intel® AVX2), Intel® AVX, SSE4.2, SSE4.1, SSE3, SSE2, SSE, and SSSE3 instructions for Intel® processors. Optimizes for Intel® processors that support Intel® AVX2 instructions.

Future work: Vectorized adding and/or multiplying

```
double A[vec_width], B[vec_width];
for(int i = 0; i < vec_width; i++)
A[i] += B[i];
```

```
double A[vec_width], B[vec_width];
__m512d A_vec = _mm512_load_pd(A);
__m512d B_vec = _mm512_load_pd(B);
A_vec = _mm512_add_pd(A_vec, B_vec);
_mm512_store_pd(A, A_vec);
```

Thank you for your attention