Co-Design Team

We are in charge of ``co-design'' of the hardware, the system software, and the application software for the post-K supercomputer.

Why we need co-design?

Modern processors are complicate system with
- Many processor, many cores, long SIMD
- Complicated memory & communication structure

Vacuum tube module of IBM 700 Series (1953-)

(c)Autopilot CC BY-SA wikipedia.org

Facom VP-100 (1982)

(c) Easy Management, Inc. Fujitsu Limited

The K computer (2012-)

Hardware makers alone...

find it difficult to make *the* general-purpose processor that execute *any* program optimally.

Programmers alone...

find it difficult to learn details of hardware features to write fast programs.

Therefore, we need to design and optimize hardware and software together.

→ That's co-design!!

That's our Mission!!

Development of Post-K computer

●Initial Phase (2014-2015):
Analyse application performance, locate bottleneck
→ Co-improvement of hardware and application software

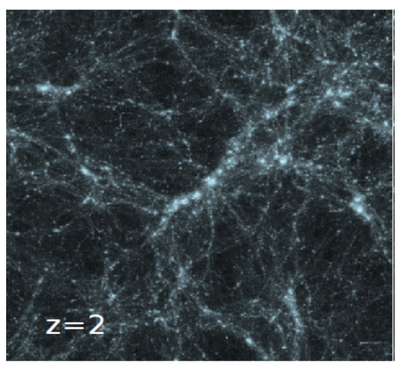
●Late Phase (2015-):
More improvement on applications

Software for co design

We design and develop application frameworks and domain specific languages (DSLs) to help HPC users implement advanced algorithms.

FDPS

... is a library for massively parallel particle simulations. Users only need to program particle interactions and do not need to parallelize the code with MPI. FDPS generates a parallel code that scales up to the K computer using highly-optimized communicationalgorithms. Now, FDPS supports GPU clusters.



Simulation of large-scale cosmic structure formation, using FDPS.

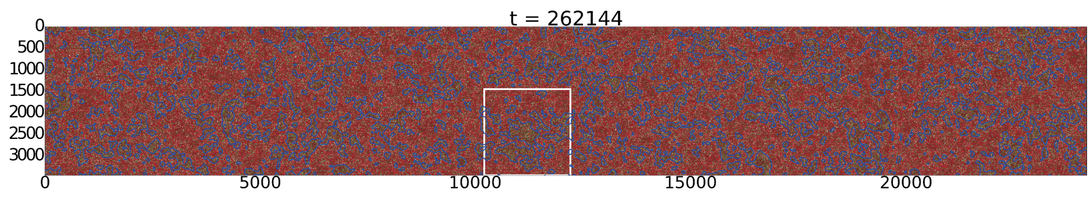
FDPS is available at https://github.com/FDPS/FDPS !!

(For more detail, see Iwasawa et al., 2016, preprint [arXiv:1601.03138])

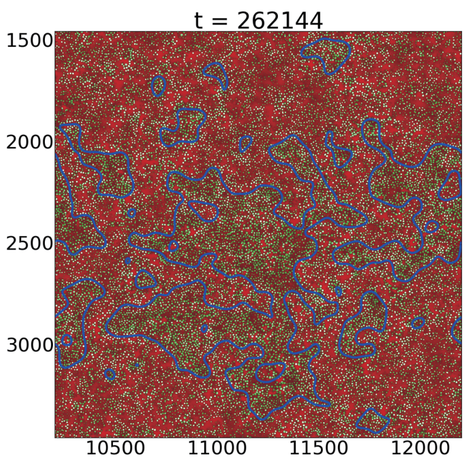
Formura

... Is a domain specific language that provides access to optimized stencil computations. Higher-order integration schemes can be defined using mathematical notations.

Formura generates C code with MPI calls, and realizes portable performance via automated tuning. Formura have been applied to magnetohydrodynamics (MHD) and belowground biology simulations. For the latter, scaling up to the full nodes of the K computer, with 1.184 Pflops, 11.62% floating-point operation efficiency, is demonstrated.



(↑) The below-ground biology simulation using Formura.
(→) Close -up of the white box
(↓) The source code for this simulation




```
dimension :: 3
axes :: x, y, z

ddx = fun(a) (a[i+1/2,j,k] - a[i-1/2,j,k])
ddy = fun(a) (a[i,j+1/2,k] - a[i,j-1/2,k])
ddz = fun(a) (a[i,j,k+1/2] - a[i,j,k-1/2])

∂ = (ddx,ddy,ddz)

Σ = fun (e) e(0) + e(1) + e(2)

begin function init() returns (U,V)
  double [] :: U = 0, V = 0
end function

begin function step(U,V) returns (U_next, V_next)
  double :: Fu = 1/86400, Fv = 6/86400, Fe = 1/900, Du = 0.1*2.3e-9, Dv = 6.1e-11
  double :: dt = 200, dx = 0.001

  double [] ::  dU_dt, dV_dt

  dU_dt = -Fe * U * V*V + Fu * (1-U) + Du/(dx*dx) * Σ fun(i) (∂ i . ∂ i) U
  dV_dt =  Fe * U * V*V - Fv * V     + Dv/(dx*dx) * Σ fun(i) (∂ i . ∂ i) V

  U_next = U + dt * dU_dt
  V_next = V + dt * dV_dt
end function
```

available at https://github.com/nushio3/formura