2026/3/11 PCCCワークショップ「アプリ開発者のためのアクセラレータプログラミング最新情報」

MN-Core HPCSDKのご紹介

Kentaro Nomura / 野村昴太郎
Engineering Manager
AIコンピューティング事業本部 ソフトウェア開発部 HPCチーム
株式会社Preferred Networks

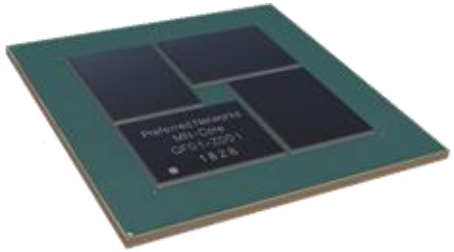
Preferred Networks

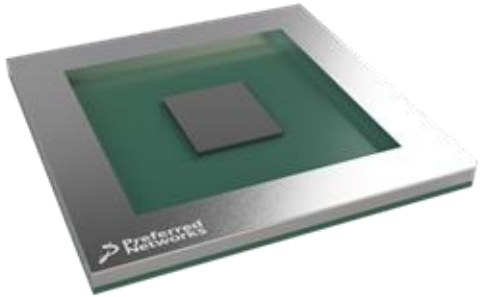
MN-Core Series

Power-efficient AI processor MN-Core™ series

AI accelerator jointly developed by Preferred Networks and Kobe University

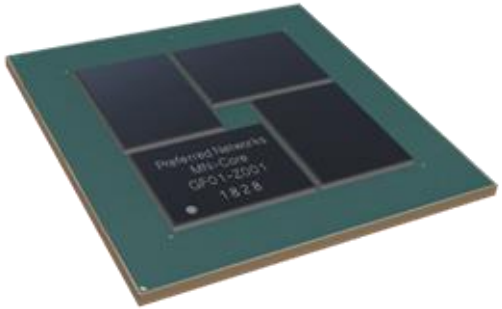
- Unique architectural design
- high performance and low energy consumption



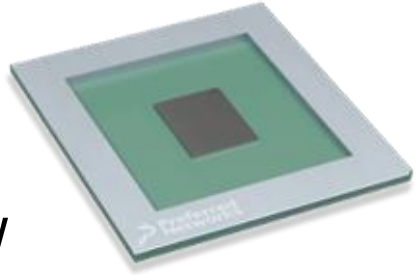


Preferred Networks

MN-Core™ Series



MN-Core
- TSMC 12nm
- 500MHz / 500W



MN-Core 2
- TSMC 7nm
- 750MHz / 322W

	MN-Core		MN-Core 2	
	Flops	GFlops/W	Flops	GFlops/W
DP(FP64)	32.8 T	66	12 T	37.24
SP(FP32)	131 T	260	49 T	148.9
quasi-SP	-	-	98 T	297.9
HP(FP16)	524 T	1000	393 T	1192

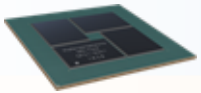
MN-Core™ Series Roadmap

2016 2020 2024 2026 2027

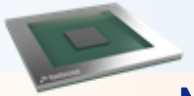
Development | Internal Use | On-Premises, Cloud

Flagship
GenAI / HPC

MN-Core
(TSMC 12nm)



MN-Core2
(TSMC 7nm)
For sale



3rd Gen.
In Development

Future Gen.
Planning

Inference

MN-Core L1000
In Development

MN-Core L2000
In Development

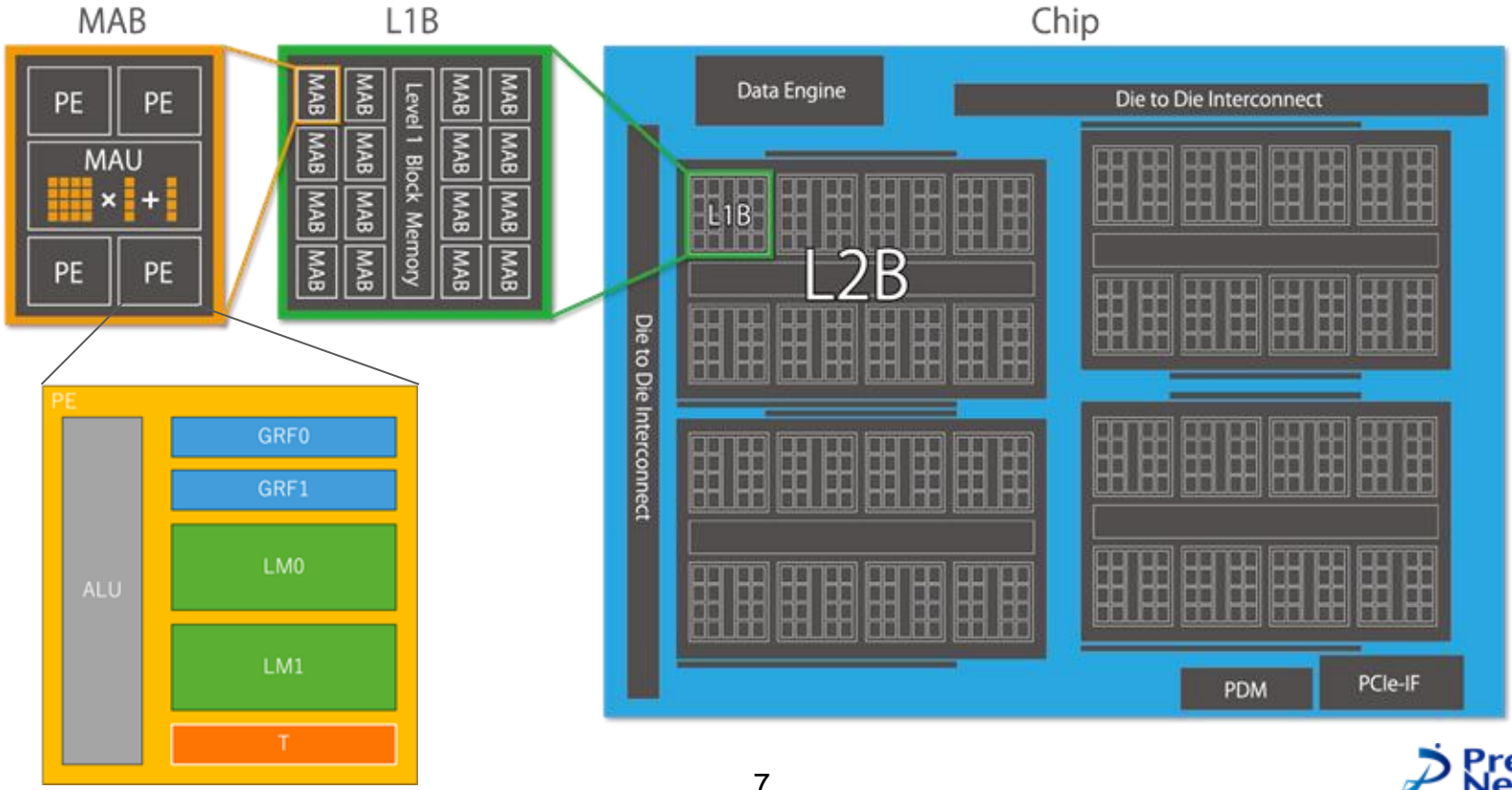
Preferred Networks

Features of MN-Core series

- Chip-wide SIMD
  - Less synchronization overhead
- Large local memories on PE
  - Near-memory computing brings better performance with relaxing DRAM access
- Hierarchical tree structure and explicit data transfer management
  - Reduction operation while data transfer
  - Overlapping computation and reduction operation

Overview of MN-Core Architecture

- MN-Core series Architecture Block Diagram

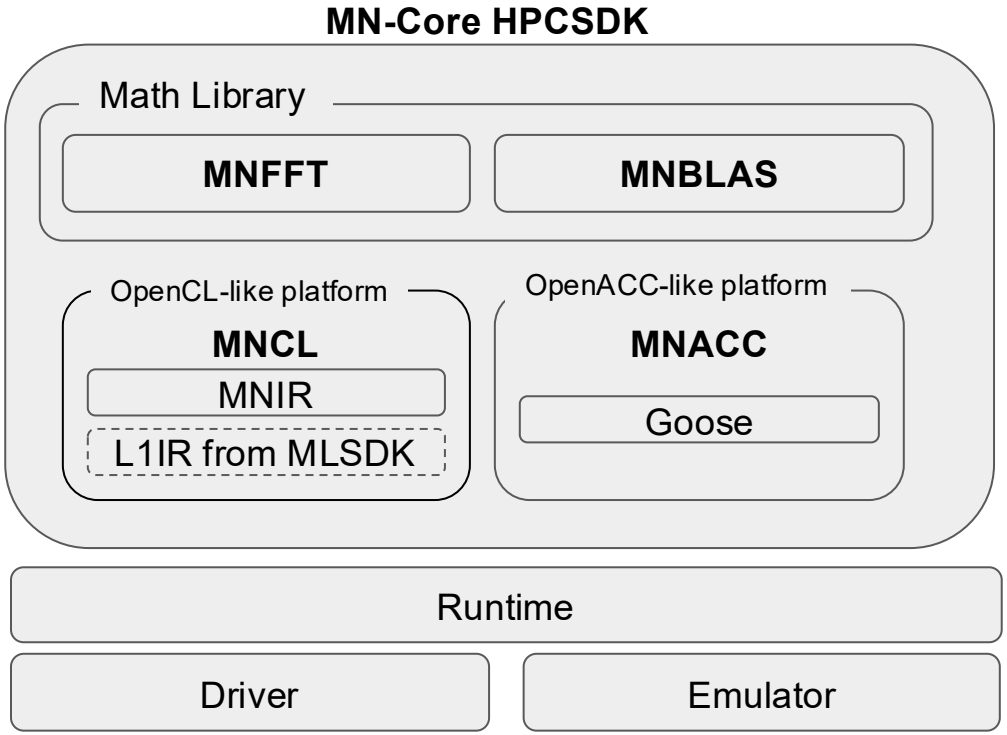


Preferred Networks

Preferred Networks

HPCSDK

MN-Core HPCSDK Overview



MN-Core HPCSDK includes:

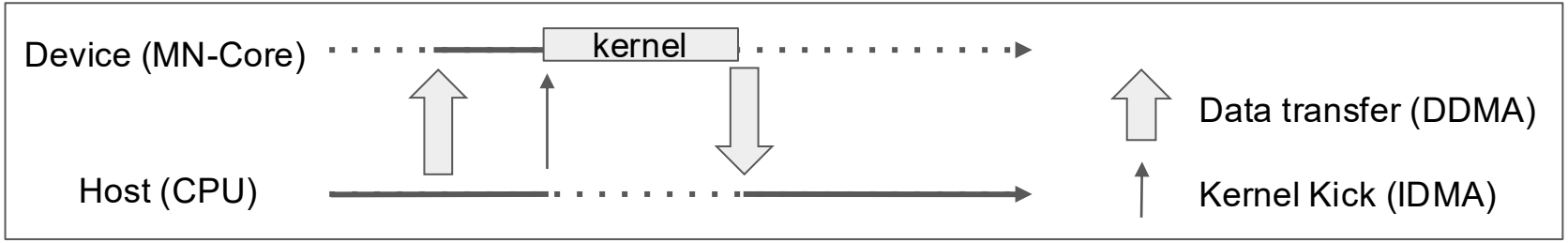
- OpenCL-like platform, **MNCL**, to develop MN-Core kernel in C++.
- OpenACC-like platform, **MNACC**, to develop directive-based MN-Core kernel.
- Math library of FFT (**MNFFT**) and BLAS (**MNBLAS**).

Preferred Networks

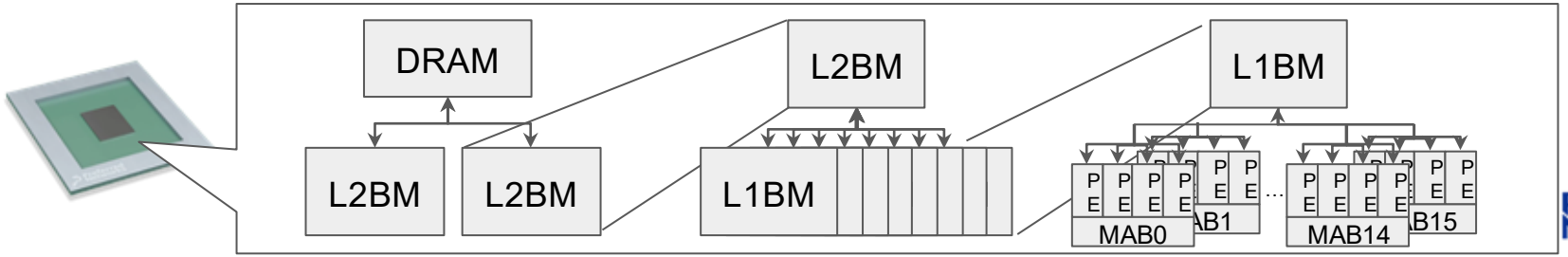
MNCL

MNCL

- Develop environment of OpenCL subset for MN-Core series.
  - MNCL provides APIs for host-device communication and kernel kick.



  - MN-Core device kernel code is written in C++.
    - Compilation and optimization using LLVM and MNIR backend.
    - Builtin functions enable simple data communication among hierarchical memory layer (DRAM <-> L2BM <-> L1BM <-> PE).



Preferred Networks

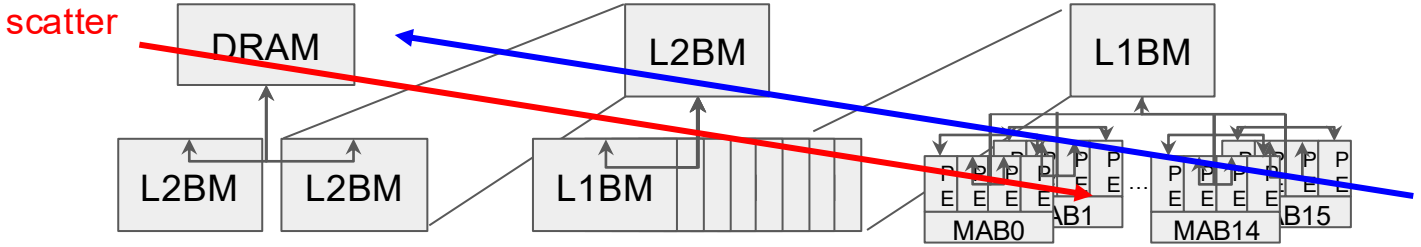
MNCL device code example of y = x + y

dev.mnclc

```
#include <mncl/device/mncore2.h>

NO_MANGLING void add(DRAM f32* __restrict__ x, DRAM f32* __restrict__ y) {
  f32 xbuf[32], ybuf[32];
  __builtin_scatter(x, (NUM_PE * sizeof(xbuf)), xbuf, sizeof(xbuf));
  __builtin_scatter(y, (NUM_PE * sizeof(ybuf)), ybuf, sizeof(ybuf));
  for(int i=0;i<32;i++) ybuf[i] += xbuf[i];
  __builtin_gather(ybuf, sizeof(ybuf), y, (NUM_PE * sizeof(ybuf)));
}
```

- Builtin function to distribute 32 x elements and 32 y elements per PE from DRAM.
- Calculate y = x+y for each element.
- Builtin function to gather results (y) to DRAM.



Preferred Networks

MNCL host code example of y = x + y (1/3)

host.cc

```
int main(){
  cl_int status;
  cl_platform_id platform;
  status = clGetPlatformIDs(1, &platform, NULL);
  cl_uint num_devices;
  cl_device_id device;
  clGetDeviceIDs(platform, CL_DEVICE_TYPE_ACCELERATOR, 1, &device, &num_devices);
  cl_context context = clCreateContext(NULL, 1, &device, NULL, NULL, &status);
  cl_command_queue command_queue = clCreateCommandQueue(context, device, 0, &status);
```

- As standard OpenCL requires, cl_context and cl_command_queue is constructed.
- Emulator can be selected by using CL_DEVICE_TYPE_EMU_MNCORE2, if you run the code without real MN-Core 2.

MNCL host code example of y = x + y (2/3)

```
size_t nelem = NUM_PE*32;
cl_mem d_x = clCreateBuffer(context, CL_MEM_READ_WRITE, nelem*sizeof(float), NULL, &status);
cl_mem d_y = clCreateBuffer(context, CL_MEM_READ_WRITE, nelem*sizeof(float), NULL, &status);

float x[nelem], y[nelem];
for(int i=0;i<nelem;i++) x[i] = (float)i;
for(int i=0;i<nelem;i++) y[i] = 0.f;
clEnqueueWriteBuffer(command_queue, d_x, CL_TRUE, 0, nelem*sizeof(float), x, NULL, NULL, NULL);
clEnqueueWriteBuffer(command_queue, d_y, CL_TRUE, 0, nelem*sizeof(float), y, NULL, NULL, NULL);
```

- Create device buffer cl_mem (allocated on DRAM) d_x and d_y
- Create host buffer x and y and initialize buffers.
- Host to device communication

MNCL host code example of y = x + y (3/3)

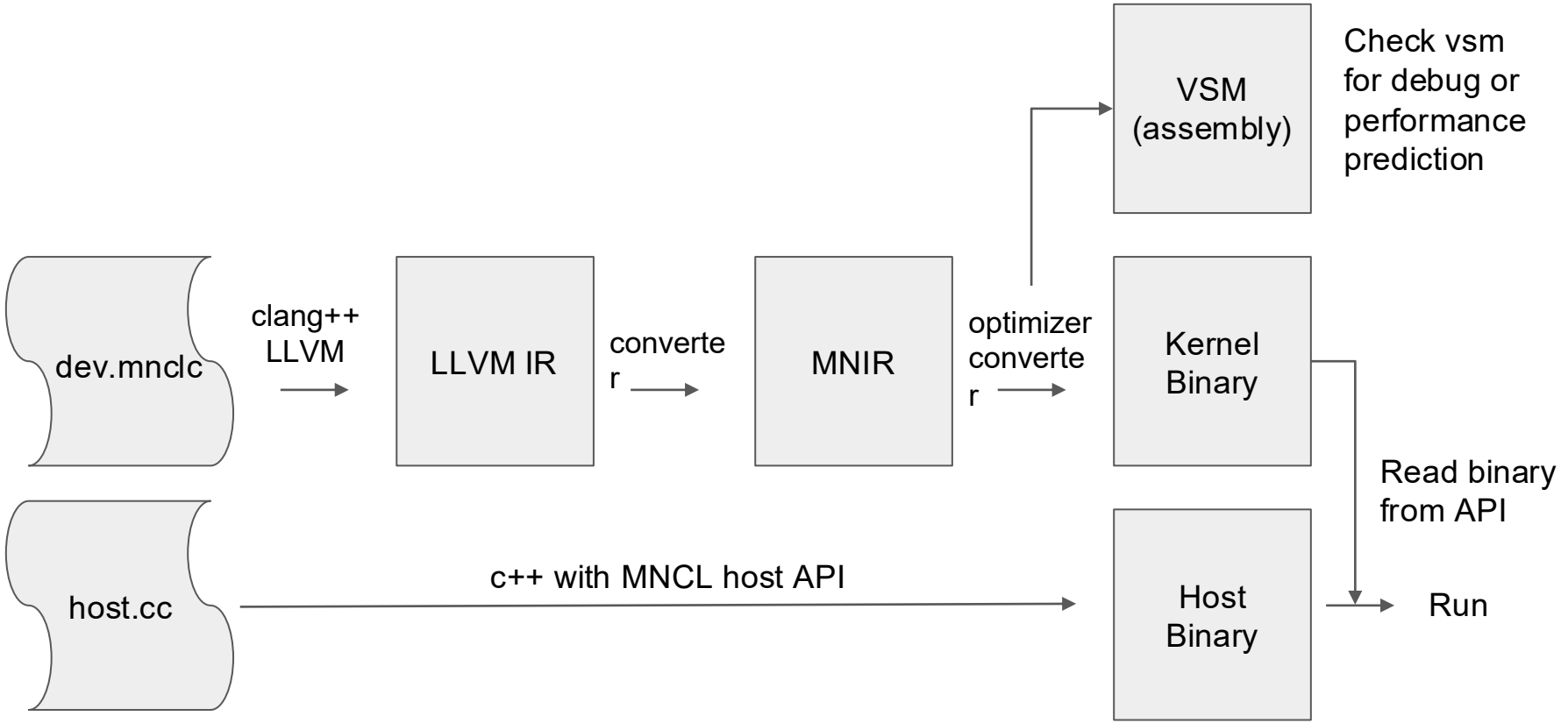
```
  raw_mncl_binary raw_binary = LoadProgram("./add.bin");
  size_t length = raw_binary.size();
  const unsigned char* ptr = raw_binary.data();
  cl_program program = clCreateProgramWithBinary(context, 1, &device, &length, &ptr, NULL, &status);
  cl_kernel kernel = clCreateKernel(program, "add", &status);
  clSetKernelArg(kernel, 0, sizeof(cl_mem), d_x);
  clSetKernelArg(kernel, 1, sizeof(cl_mem), d_y);

  clEnqueueTask(command_queue, kernel, 0, NULL, NULL);

  clEnqueueReadBuffer(command_queue, d_y, CL_TRUE, 0, nelem*sizeof(float), y, NULL, NULL, NULL);
  for(int i=0;i<nelem;i++) assert(x[i] == y[i]);
  /* releasing resources is omitted */
} // end of main
```

- Reading compiled kernel code binary and create cl_prgoram and cl_kernel.
- Set d_x and d_y as kernel arguments of `add(DRAM f32* x, DRAM f32* y)`.
- Enqueue kernel to device.
- Read results from device and check the retrieved results.

MNCL compilation flow



Preferred Networks

Preferred Networks

MNACC

MNACC

- OpenACC-like develop environment
  - Directive-based kernel programming.
  - Automatic data mapping for DRAM, local memories and registers.

```
#pragma mnacc parallel for loopcounter(i,j) result(c)
for(int i=0;i<ni;i++) {
    c[i] = 0.0;
    for(int j=0;j<nj;j++) {
        c[i]+=a[i][j]*b[j];
    }
}
```

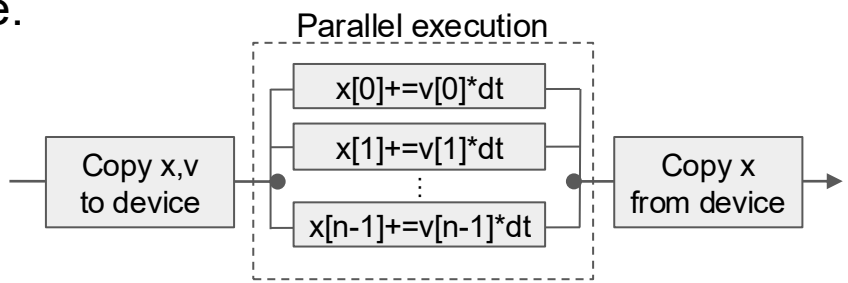
Supported MNACC Pragmas

- **mnacc config ni_loop(ni) (nj_loop(nj) (nk_loop(nk))?)?**
  - Declares the number of iterations of first (, second and third) loop of the nested loops.
- **mnacc data first_last**
  - Declares that the host-device data transfer is required at the first and last iteration of following loop.
- **mnacc parallel for loopcounter(vars0…) result(vars1…)**
  - Declares that the following loop is a device kernel region with specifying loop counter variables by loopcounter clause and result values and pointers by result clause.
- **mnacc kernels copyout(vars…)**
  - Also declares thet the following loop is a device kernel region with specifying result values and pointers by copyout clause.
- **mnacc loop independent**
  - Tells the implementation that the loop iterations must be data independent

#pragma mnacc parallel for loopcounter(var) result(vars…)

```
#pragma mnacc parallel for loopcounter(i) result(x)
for(int i=0;i<ni;i++) {
  x[i]+= v[i]*dt;
}
```

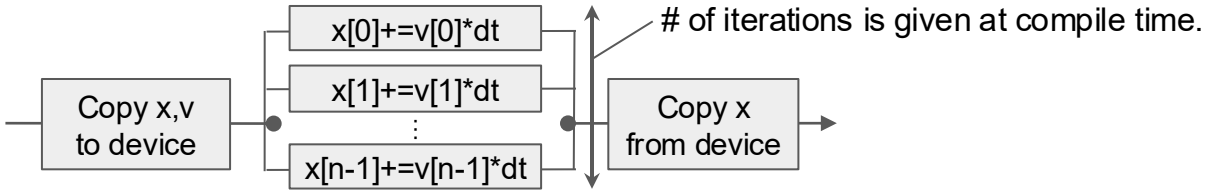
- The **parallel** construct declares to start the parallel execution on the device and, for now, must be followed by the **for** construct.
- The **loopcounter** clause declares the counter variable(s) of the target loop(s).
- The **result** clause declares the variable or pointer to be updated on device and retrieved from device.



#pragma mnacc config ni_loop(var)

```
constexpr int NI = 1024;
#pragma mnacc config ni_loop(NI)
#pragma mnacc parallel for loopcounter(i) result(x)
for(int i=0;i<NI;i++) {
  x[i]+= v[i]*dt;
}
```

- The **config** construct provides compiler information for compile time optimization.
- The **ni_loop** clause proides the number of iterations of i loop at compile time.

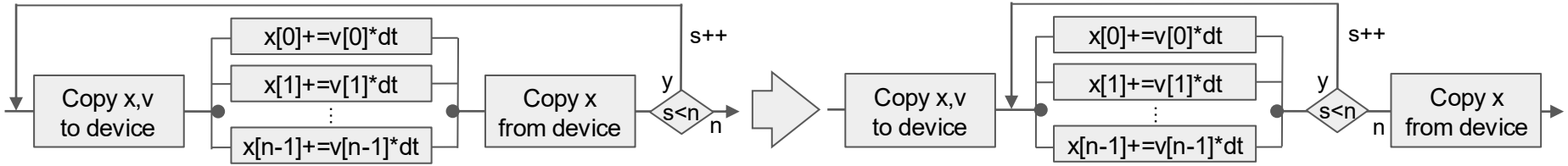


Preferred Networks

#pragma mnacc data first_last

```
#pragma mnacc data first_last
for(int s=0;s<n;s++){
  #pragma mnacc parallel for loopcounter(i) result(x)
  for(int i=0;i<ni;i++) {
    x[i]+= v[i]*dt;
  }
}
```

- The **data** construct defines the variables to be allocated in the device memory for the duration of the region and whether data transfer to/from device is performed.
- The **first_last** clause declares that the data transfer to/from the device is performed only at first/last iteration of the loop.

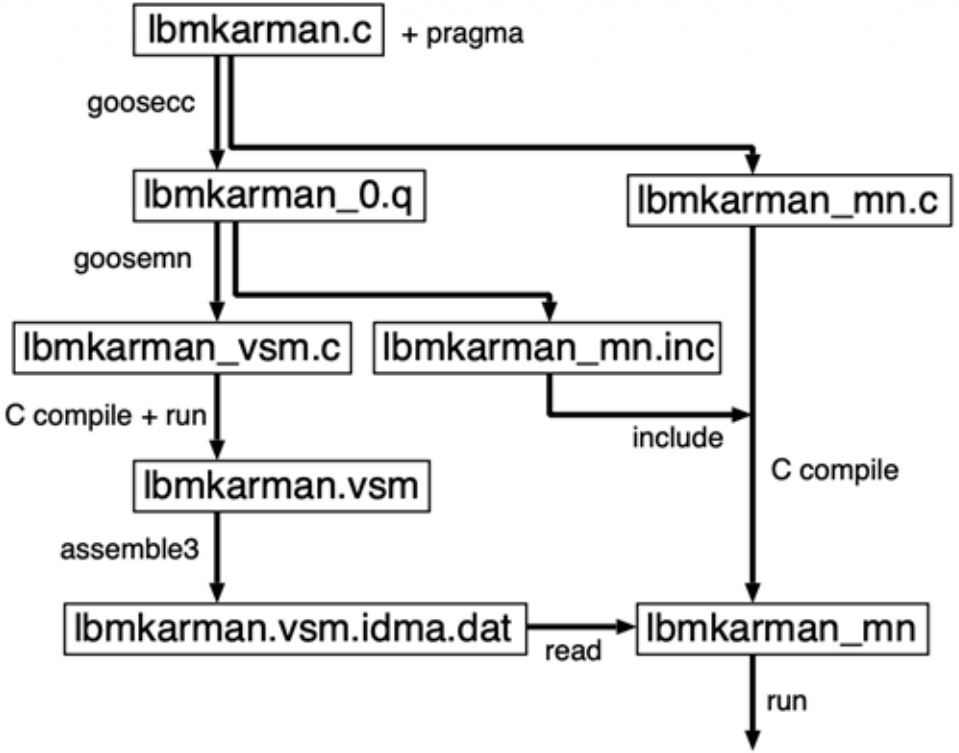


without data first_last

with data first_last

Preferred Networks

Compilation flow



- Source code written in C and pragma is translated and divided into intermediate file (.q) and host code (_mn.c) by goosecc.
- .q file is translated into program to generate vsm (assembly) (_vsm.c) and include file (.inc) which provides functions for initialization and host-device instruction/data transfer.
- vsm is translated into device kernel binary (.vsm.idma.dat).
- The host code is compiled to binary and load the device kernel at runtime.

Preferred Networks

MN-Core Math Library

Preferred Networks

MNBLAS

MNBLAS Implementation Status

Level 1: vector

(s|d|c|z)axpy
(s|d|c|z|cs|zd)scal
(s|d|c|z)copy
(s|d|c|z)swap
(s|d|sds|ds)dot
(c|z)dotu
(c|z)dotc
(s|d|sc|dz)nrm2
(s|d|sc|dz)asum
(s|d|c|z)i_amax
(s|d|c|z)rotg
(s|d|c|z|cs|dz)rot
(s|d)rotmg
(s|d)rotm

Level 2: matrix-vector

(s|d|c|z)gemv
(c|z)hemv
(s|d)symv
(s|d|c|z)trmv
(s|d|c|z)trsv
(s|d)ger
(c|z)geru
(c|z)gerc
(s|d|syr
(c|z)her
(s|d)syr2
(c|z)her2

Level 2: band storage

(s|d|c|z)gbmv
(c|z)hbmv
(s|d)sbmv
(s|d|c|z)tbmv
(s|d|c|z)tbsv

Level 2: packed storage

(c|z)hpmv
(s|d)spmv
(s|d|c|z)tpmv
(s|d|c|z)tpsv
(s|d)spr
(c|z)hpr
(s|d)spr2
(c|z)hpr2

Level 3: matrix-matrix

(s|d|c|z)gemm
(s|d|c|z)gemmtr
(s|d|c|z)symm
(c|z)hemm
(s|d|c|z)trmm
(s|d|c|z)trsm
(s|d|c|z)syrk
(c|z)herk
(s|d|c|z)syr2k
(c|z)her2k

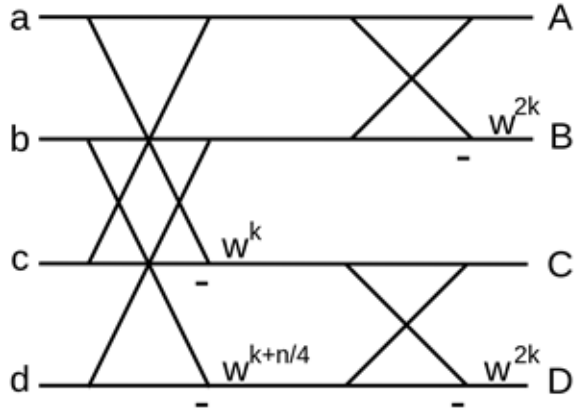
Optimized
Implemented (not optimized)
Partially implemented
Not implemented

Preferred Networks

MNFFT

MNFFT

- MNFFT is a high performance library to compute discrete Fourier Transform (DFT) in one or more dimensions for MN-Core series.
- Decimation-in-Frequency Cooley-Turkey FFT is adopted to align with chip-wide SIMD and to fully utilize matrix arithmetic units (MAU).
- The interface is a subset of clFFT.



$$\begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} = \begin{pmatrix} w^0 & w^0 & w^0 & w^0 \\ w^{2k} & -w^{2k} & w^{2k} & -w^{2k} \\ w^k & w^{k+n/4} & -w^k & -w^{k+n/4} \\ w^{3k} & -w^{3k+n/4} & -w^{3k} & w^{3k+n/4} \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$$

Efficiently computed on MAU!

Preferred Networks

Preferred Networks

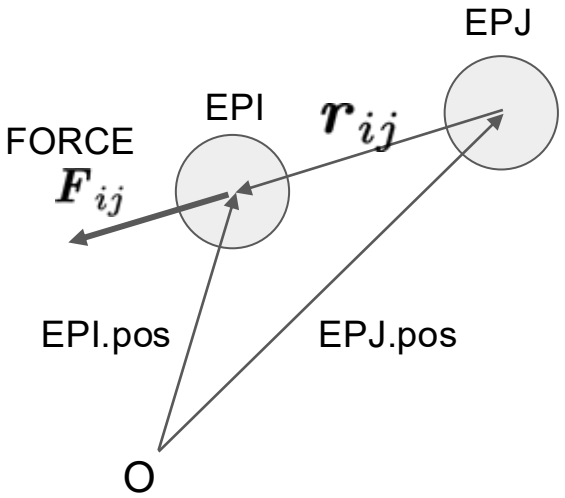
HPC Software on MN-Core

PIKG (Particle-particle Interaction Kernel Generator)

- A DSL to generate interaction kernel of particle-based simulation, such as N-body and MD simulation, for various architecture from single source.
  - Supports AVX2, AVX-512, ARM SVE, CUDA, **MN-Core 2 VSM**.

Lennard-Jones atom example

$$\boldsymbol{F}_{ij} = \epsilon \left\{ 48 \left(\frac{\sigma}{|\boldsymbol{r}_{ij}|} \right)^{12} - 24 \left(\frac{\sigma}{|\boldsymbol{r}_{ij}|} \right)^{6} \right\} \frac{\boldsymbol{r}_{ij}}{|\boldsymbol{r}_{ij}|^2}$$



Sample code

```
# σ = ε = 1.0
rij = EPI.pos - EPJ.pos
r2 = rij * rij
rinv = rsqrt(r2)
rinv2 = rinv * rinv
rinv6 = rinv2 * rinv2 * rinv2
f = rinv6 * rinv2 * (48.0f * rinv6 - 24.0f)
FORCE.force -= to_f64vec(rij) * to_f64(f)
```

Available on https://github.com/FDPS/PIKG

Preferred Networks

製品紹介

MN-Core 2 Devkit

MN-Core 2ボードを1枚搭載するデスクトップマシン。

搭載AIアクセラレータ/数：MN-Core 2 / 1枚

AIアクセラレータ総理論演算性能：TF16 393TF

標準販売価格：200万円(税抜)

MN-Server 2

MN-Core 2ボードを8枚搭載するラックマウント型5Uサーバ。

型番：MNS2V1

搭載AIアクセラレータ/数：MN-Core 2 / 8枚

AIアクセラレータ総理論演算性能：TF16 3.1PF

標準販売価格：2000万円(税抜)

https://projects.preferred.jp/mn-core/assets/MN-Core2-hardware-catalog.pdf

Preferred Networks

Preferred Networks

Making the real world computable