MN-Core™ 2 White Paper

2023-11-12

Preferred Networks

Introduction

AI and HPC workloads demand ever-increasing amounts of computational resources. In addition, models such as large-language models (LLMs), which are much more complex and large-scale than conventional models and require huge amounts of computational resources, are being incorporated into actual products and used in the real world. To meet these computational demands, an ultra-fast and ultra-energy-efficient accelerator for AI is the key. Preferred Networks has developed a series of accelerators called MN-Core series.

The first generation accelerator, MN-Core, has demonstrated that its architecture is extremely power efficient, and MN-Core has performed extremely well in the Green500 list. Below is a table showing the performance per power consumption (GFlops/W) of MN-Core and its ranking at that time. It has also proven to significantly outperform existing GPUs on a variety of real-world AI workloads.

	Energy Efficiency	Rank
Jun. 2020	21.11 GFlops/W	#1
Nov. 2020	26.04 GFlops/W	#2
Jun. 2021	29.70 GFlops/W	#1
Nov. 2021	39.38 GFlops/W	#1

MN-Core 2 is the second generation of accelerators developed by Preferred Networks and, as described in this document, offers increased memory bandwidth and a significant cost reduction compared to the first generation MN-Core. In addition to software for AI workloads, MN-Core 2 comes with a general-purpose development environment and supports general HPC workloads other than AI. This enables the high computing performance of MN-Core 2 to be utilized in a wide variety of existing HPC applications.

MN-Core 2 Overview

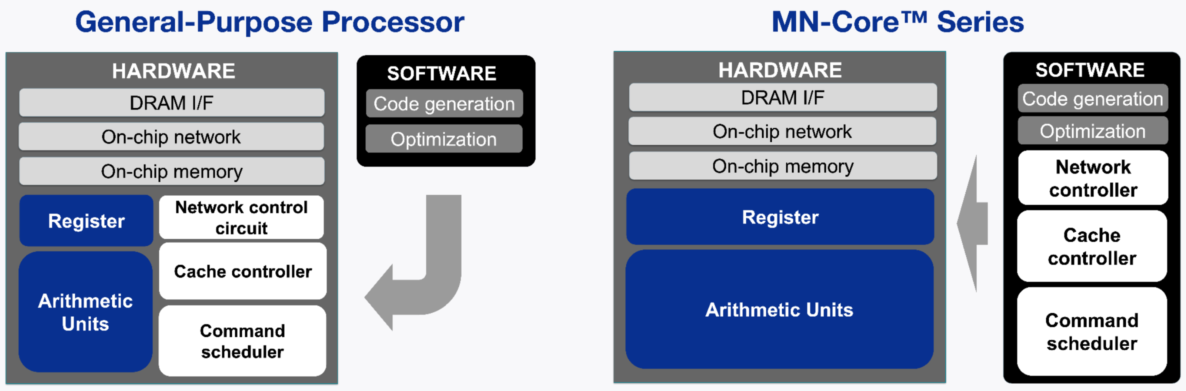
About MN-Core 2

MN-Core 2 is the second generation of the MN-Core series of accelerators developed by Preferred Networks.The MN-Core series has the following features

- High silicon utilization efficiency
- Achieve high execution efficiency through software optimization

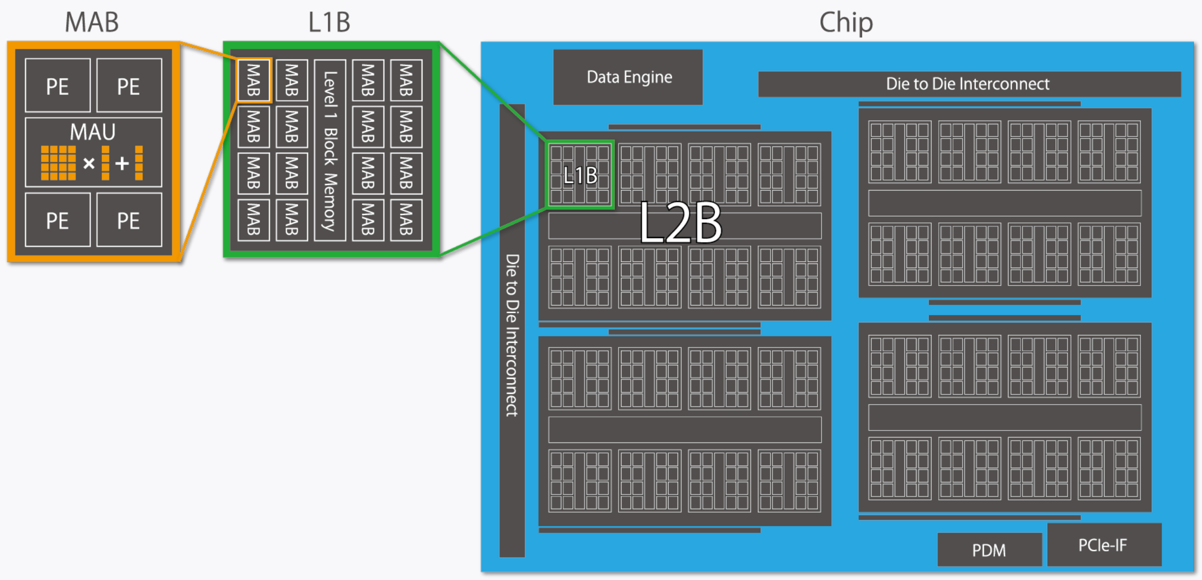
Many of today's processors share the same characteristic that the fraction of the area occupied by the arithmetic unit on the actual silicon is very low. This can be translated as ample hardware support to ease users' architectural migration by minimizing the amount of modification to the existing code base. On the other hand, the MN-Core series is designed to achieve a very high ratio of the area for arithmetic units to the total silicon area. The following table illustrates the difference in design philosophy between the existing processors and the MN-Core series. For MN-Core 2, the ratio of the number of transistors for arithmetic units to the total number of transistors is about 7 %. This is a very high figure compared to processors from other companies. (Figures are based on Preferred Networks' own research.)

Device Name	ratio of the number of transistors for arithmetic units
MN-Core 2	7.4 %
GPU N	1.7 %
Accelerator P	2.4 %
CPU F	1.3 %
CPU I	0.8 %



The MN-Core series has an architecture that minimizes the amount of hardware control logic on the silicon, so software optimization is extremely important. Processing Element (PE) on the accelerator does not have its own program counter or instruction decoder. All PEs operate in perfect synchronization, receiving instruction sequences generated by the host CPU directly from the host. This eliminates the load-imbalance and associated synchronization costs that often occur on today's accelerators due to the asynchronous operation of the individual computation units on the accelerator, and also eliminates bottlenecks in the instruction supply system, such as instruction caches, and and efficiency degradation due to lack of, for example, out-of-order resources.

Below is the MN-Core series architecture diagram.



The smallest unit is a PE, with 4 PEs sharing a single MAU to form an MAB, 16 MABs form an L1B, 8 L1Bs form an L2B.

The table below shows the specifications of MN-Core 2.

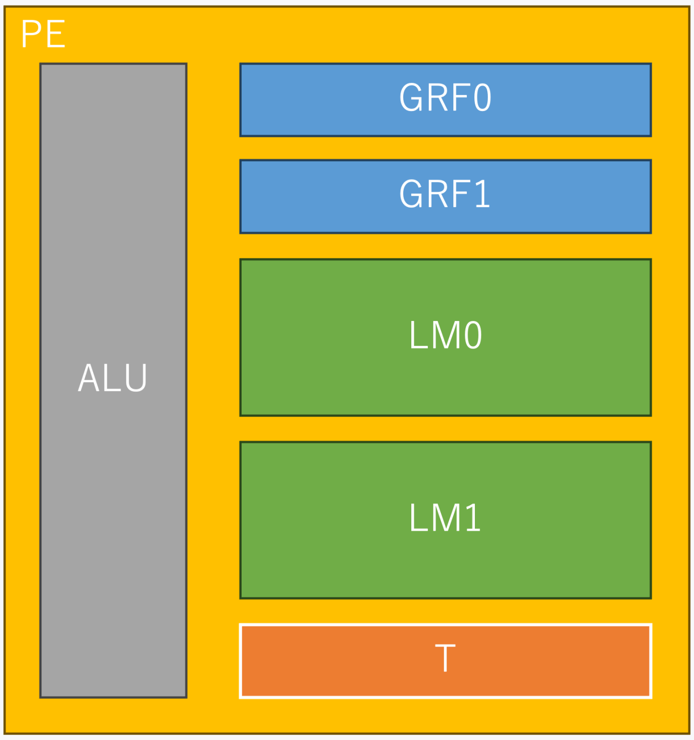
Manufacturing Process	TSMC N7
Area	550 mm2
Number of Transistors	22 B
Operating Frequency	750 MHz
Number of PEs	4096
Peak FLOPS @ fp64	12 TFlops
Peak FLOPS @ fp32	49 TFlops
Peak FLOPS @ TF32	98 TFlops
Peak FLOPS @ TF16	393 TFlops
Power Consumption	330 W (Design value)
Energy efficiency (fp64)	37.24 GFlops/W
Energy efficiency (fp32)	148.9 GFlops/W
Energy efficiency (TF32)	297.9 GFlops/W
Energy efficiency (TF16)	1192 GFlops/W

The following sections describe the architecture of MN-Core 2.

Processing Element

The Processing Element (PE) has a General Register File (GRF) and Local Memory (LM) to hold data. Data read from these memories is input to ALUs and other arithmetic units. The output of the calculation results are stored in these memories. In addition to the memories, the PE has a T register for holding temporary data and mask registers for holding operation result flags. PSs can perform general operations as well as special operations such as ReLU. PEs can also exchange data with the upper hierarchical memory L1B.

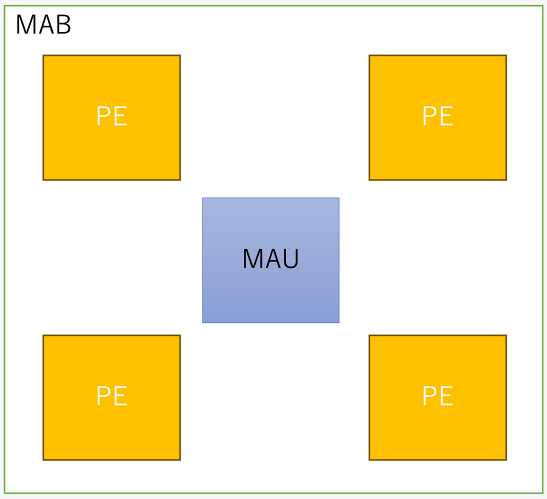
The diagram below shows the architecture of a single PE.



MAB / MAU

The Matrix Arithmetic Block (MAB) consists of four PEs and one Matrix Arithmetic Unit (MAU).

The diagram below shows the architecture of a MAB.

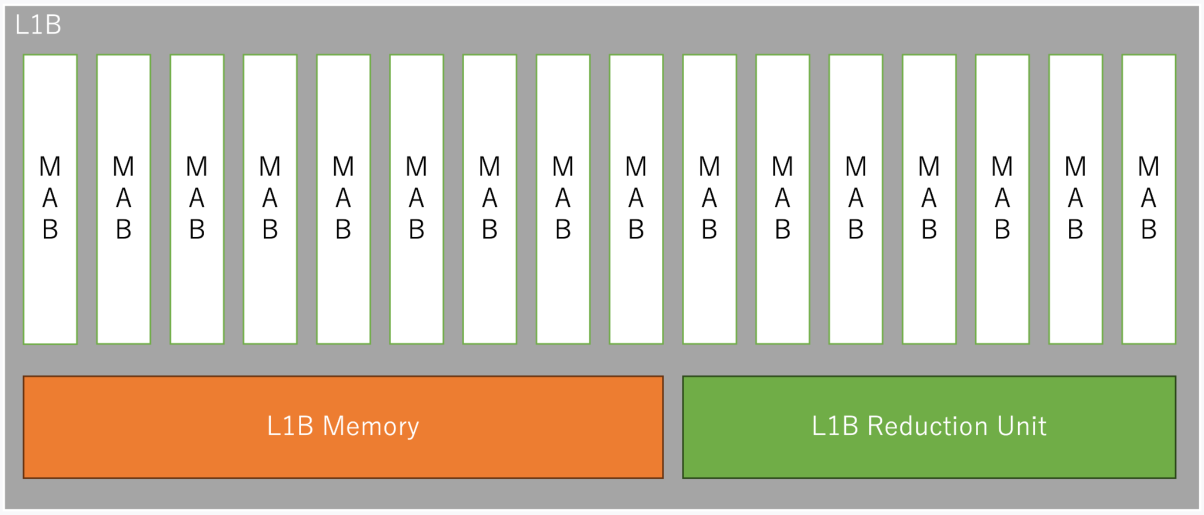


The MAU receives input data from the PEs, performs arithmetic operations, and returns output data to the PEs. The MAU also has a dedicated matrix register used for matrix operations, which holds the matrix used in the matrix-vector product. Since the matrix register has two sides, it is possible to perform matrix operations using matrix register 2 while writing to matrix register 1.

The MAU can perform the following operations:

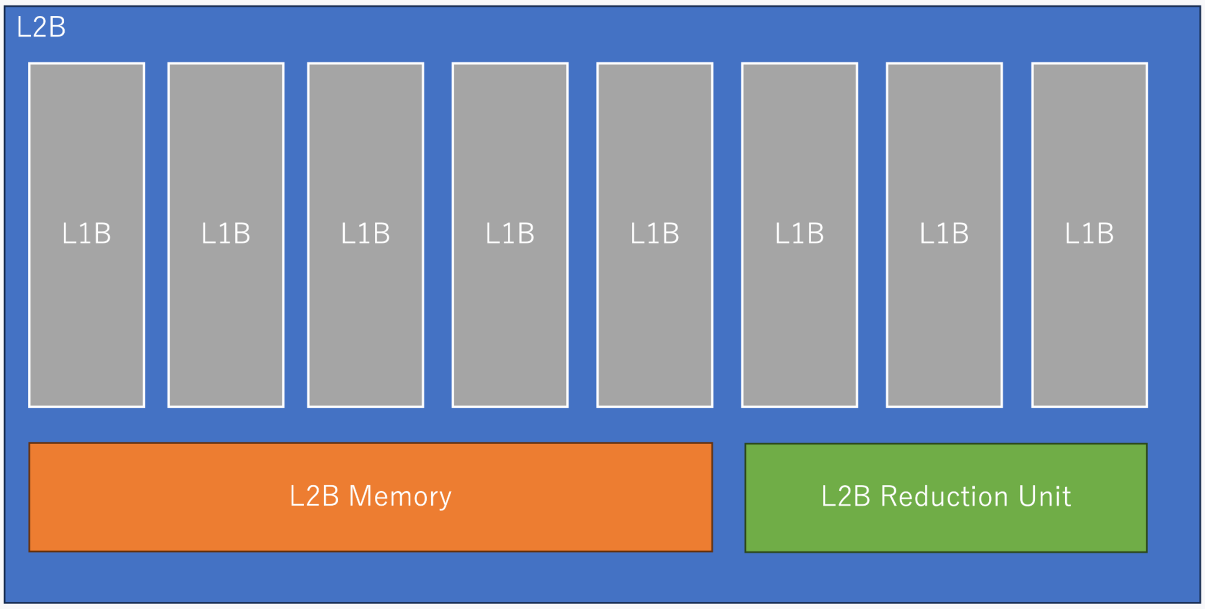
- Fused Multiply-Add (FMA) operations
    - Vector FMA (A x B + C) : half, single and double floating point precision.
    - Matrix FMA (A x matrix register + C): half, pseudo single, single and double floating point precision.
- Write to a matrix register: half, pseudo single, single and double floating point precision.
- Transpose read from to a matrix register: half, pseudo single, single and double floating point precision.

L1B



.
The Level-1 Broadcast Block (L1B) has 16 MABs and one L1 Broadcast Memory (L1BM). L1BM and MABs can perform a variety of data transfers, including broadcast data transfer from L1BM to MABs (or PEs), individual and data transfer, reduction operation, and distributing/combining data transfer. These data transfers are controlled by PE instructions. This variety of transfer modes enables highly efficient execution of fine-grained parallel tasks, which is difficult with multiprocessors using hierarchical caches.

L2B



The Level-2 Broadcast Block (L2B) contains eight L1Bs and one L2 Broadcast Memory (L2BM). As was the case for between L1BM and PE/MAB, L2BM and L1B can perform a variety of data transfers such as broadcast data transfer from L2BM to L1B, individual data transfers, reduction data transfers and distributing/combining data transfers. These data transfers are also controlled by the PE instruction. Finally, the L2B can transfer data to the host interface and external DRAM in a variety of ways.

MN-Core 2 Equipped Server MN-Server 2

MN-Server 2 is equipped with 8 MN-Core 2 boards. The specifications are as follows.

CPU	Intel Xeon Platinum 8480+ x2
RAM	1 TB
Storage	960 GB(System) + 45 TB(Data)
NIC	NVIDIA ConnectX-6 100GbE Ethernet Adapter Dual port x2
Accelerator	MN-Core 2 x8

While the first-generation MN-Core equipped server was 7U, the MN-Server 2 is now 5U, effectively reducing the height and increasing the number of servers that can be mounted on a 19-inch rack. The configuration of the MN-Server 2 mounted on a 19-inch 42U rack is called an MN-Pod. The computational performance per MN-Pod is 2.25 times that of the first-generation MN-Core when comparing using half-precision floating-point numbers.

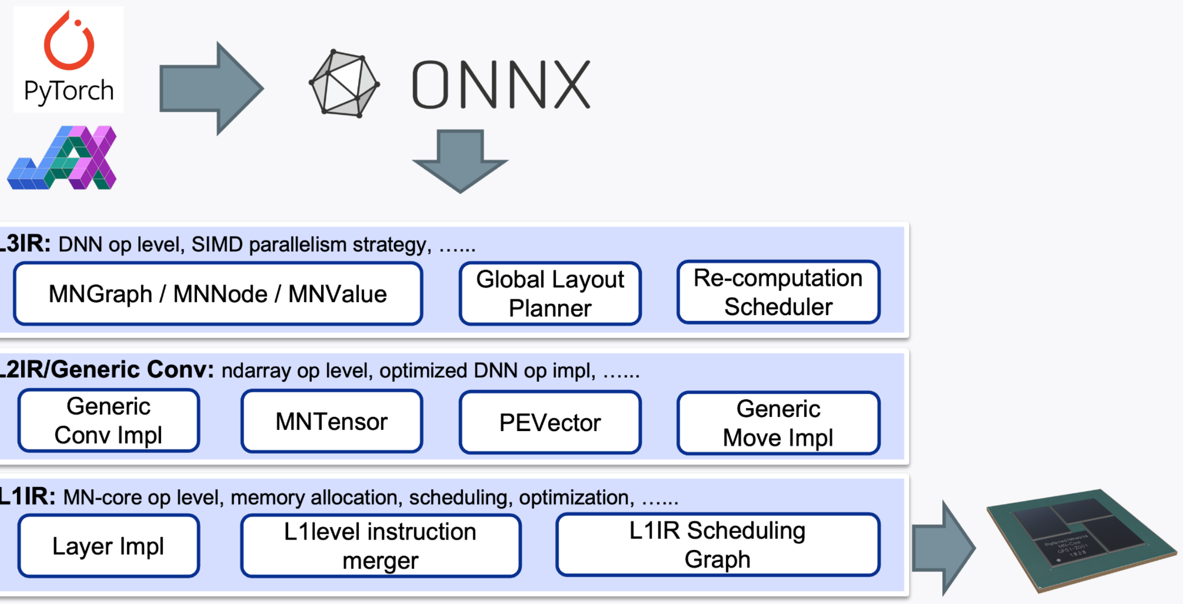
	MN-Pod	MN-Pod 2	Intergenerational performance changes
Double-precision floating-point (TFlops)	524.29	590	1.125
Single-precision floating-point (TFlops)	2097.15	2359	1.125
Pseudo-Single-precision floating-point (TFlops)	N/A	4719	N/A
Half-precision floating-point (TFlops)	8388.61	18874	2.25

MN-Core 2 is designed with the user's needs in mind and will be provided in various models such as on-premises, IaaS, and SaaS.

Software Stack

MN-Core AI Software Stack

The MN-Core AI software stack has the structure shown below.


PyTorch
ONNX
L3IR: DNN op level, SIMD parallelism strategy, ...
MNGraph / MNNode / MNValue
Global Layout Planner
Re-computation Scheduler
L2IR/Generic Conv: ndarray op level, optimized DNN op impl, ...
Generic Conv Impl
MNTensor
PEVector
Generic Move Impl
L1IR: MN-core op level, memory allocation, scheduling, optimization, ...
Layer Impl
L1level instruction merger
L1IR Scheduling Graph

In designing the AI software stack, we focus on the following two points

- To reduce major modifications of user code as much as possible
- To exploit the performance of the MN-Core series.

Today, many users use PyTorch as a framework for deep learning, and many R&D assets are based on PyTorch. Therefore, it is desirable for the MN-Core series to have as few modifications to existing PyTorch code as possible. To satisfy this, ONNX is used as input for the MN-Core AI software stack. Models expressed in ONNX can be run on MN-Core, albeit with some restrictions.

Since the MN-Core series is a highly powerful SIMD processor, it is essential to effectively supply each computation core with the appropriate data in order to unleash its performance. To achieve optimal performance, it is necessary to map the required instructions and data appropriately to the PE, MAB, L1B, and L2B components. However, it is generally challenging for users to deeply understand these hardware structures and write suitable code. Therefore, in the MN-Core series, especially in the context of AI-oriented software stacks, an automated mechanism is implemented to handle everything from data structure to code generation. This enables users to obtain the high performance of the MN-Core series without having to be conscious of hardware intricacies such as memory layout and instruction issuing.

The following is a code written in PyTorch that executes ResNet50 on a CPU.

Python
```python
import torch
import torchvision
import datasets

dataset = datasets.ImagenetDataset()
model = torchvision.models.resnet50(pretrained=True)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
loader = torch.utils.data.DataLoader(dataset, batch_size=128)

def model_with_loss(inputs, labels):
  y = model(inputs)
  return criterion(y, labels)

for epoch in range(10):
  for idx, (inputs, labels) in enumerate(loader):
    optimizer.zero_grad()
    loss = train_step(inputs, labels)
    loss.backward()
    optimizer.step()
    print(f"iter {idx}: {loss}")
```

Changing the code to be executed by MN-Core results as follows.

```python
import torch
import mncore
import torchvision
import datasets

dataset = datasets.ImagenetDataset()
model = torchvision.models.resnet50(pretrained=True)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
loader = torch.utils.data.DataLoader(dataset, batch_size=128)

def model_with_loss(inputs, labels):
  y = model(inputs)
  return criterion(y, labels)

train_step = mncore.compile(model_with_loss, backward=True,
optimizer=optimizer)

for epoch in range(10):
 for idx, (inputs, labels) in enumerate(loader):
   # Compiled model for MN-Core executes Forward, Backward and Optimizer
Steps together.
   loss = train_step(inputs, labels)
   print(f"iter {idx}: {loss.cpu()}")
```

MN-Core General Purpose Software Stack

This section describes the general purpose software stack for MN-Core 2.

MN-Core 2 is targeted to process AI workloads at very high speeds, but it can likewise leverage its computing power in some general-purpose computations. OpenACC and OpenCL, subsets of each, are under development as software stacks for general-purpose computation.

OpenACC for MN-Core

OpenACC is a parallel computing framework proposed and standardized by the OpenACC Organization. It was designed to simplify parallel programming of heterogeneous systems. For more information, visit https://www.openacc.org.

MN-Core 2 supports a subset of OpenACC. It will be possible to run the codes like the following one on MN-Core 2.

```
C/C++
void vecadd(...) {
  ...
  #pragma acc data copyin(a[0:1024], b[0:1024]) copyout(c[0:1024]) l2(a,b,c[8])
l1(a,b,c[1])
  {
  #pragma acc parallel
  #pragma acc loop independent
    for(int i=0; i<1024; i++) {
      c[i] = a[i]+b[i];
    }
  }
}
```

OpenACC support is currently under development.

OpenCL for MN-Core

OpenCL is a cross-platform API for parallel computing, proposed and standardized by the Khronos Group. It enables fine-tuned programming of hardware than OpenACC does. For more information, see https://www.khronos.org/opencl/.

MN-Core 2 supports a subset of OpenCL, allowing code running on MN-Core 2 to be written in a form similar to the OpenCL C language.

OpenCL support is currently under development.

Conclusion

This document describes MN-Core 2, a second-generation accelerator developed by Preferred Networks.

MN-Core 2 is a very fast and low-power accelerator, and MN-Core 2, MN-Server 2, and MN-Pod 2, which integrates MN-Core 2, have achieved very high performance per area. This performance is revolutionary.

MN-Core 2 will be available in a variety of ways, including installation on customer workstations, operation in customer data centers, and as a cloud service (IaaS, SaaS, etc.). MN-Core 2 will help accelerate your workloads, whether they are AI or HPC workloads.

https://projects.preferred.jp/mn-core/

Revision History

- 2023-11-10 ver 0.1 Initial version