IWAHPCE, 27th February, 2023

OpenACC Execution Models for Manycore Processor with ARM SVE

Mitsuhisa Sato Team Leader of Programming Environment Team

RIKEN Center for Computational Science (R-CCS)

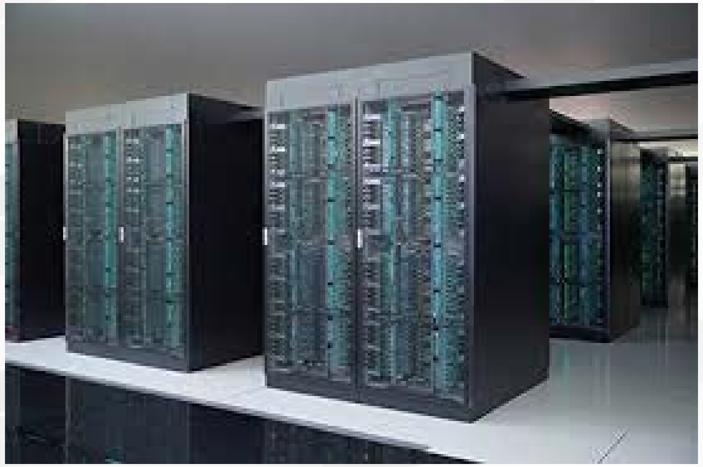
Professor (Cooperative Graduate School Program), University of Tsukuba

Miwako Tsuji, R-CCS

RIKEN R-CCS

Supercomputer “Fugaku” and A64FX processor

- **Ultra-scale “general-purpose” manycore system: 158,976 nodes (1 processor/node, total 7.6 M cores, theoretical peek 537PFLOPS (DP))**
- **Arm-based manycore processor: Fujitsu A64FX (Armv8.2-A SVE 512bit SIMD, #core 48 + 2/4, 3TF@2.0GHz, boost to 2.2GHz)**
  - 12 cores in a cluster of cores called CMG, connected to L2 and HBM memory chips
- **Advanced Memory technology: HBM2 32 GiB, 1024 GB/s bandwidth, packaged in CPU chip**
- **Scalable Interconnect: ToFu-D interconnect**

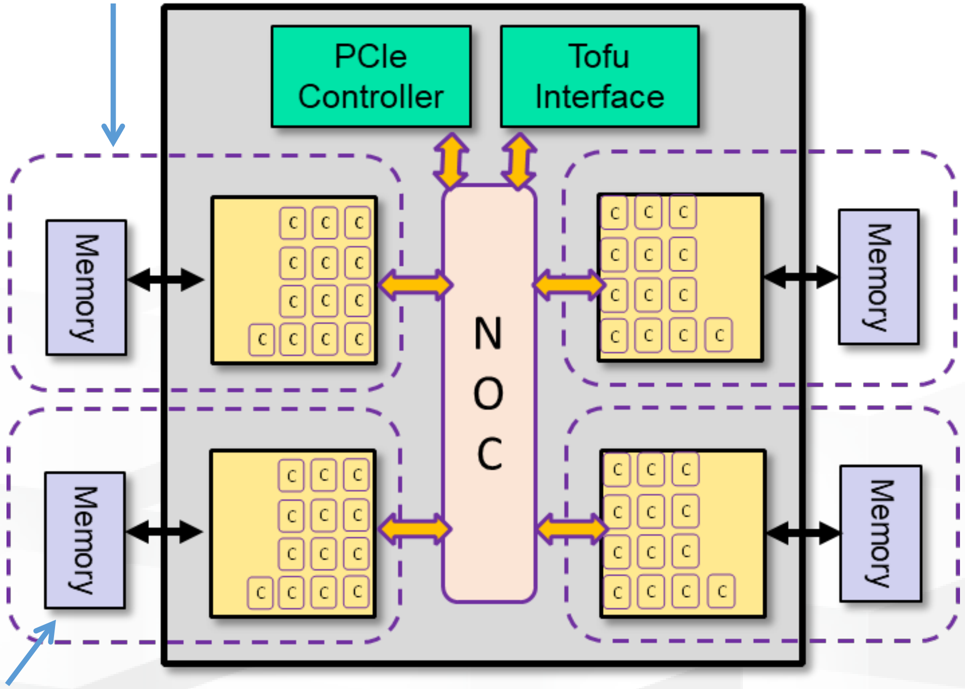


◆ Standard programing model is OpenMP-MPI hybrid programming. running each MPI process on a NUMA node (CMG).

◆ 48 threads OpenMP is also supported.

Network interface and PCIe are integrated

CMG(Core-Memory-Group): NUMA node
12+1 core



HBM2: 8GiB

Diagram of A64FX processor

Motivation and Objectives of OpenACC for A64FX

- **OpenACC as a Programming model to exploit parallelism of A64FX architecture**
  - HPC oriented design of 64FX
  - GPU-like code generation may improve the performance?

Performance Characteristics for A64FX processor

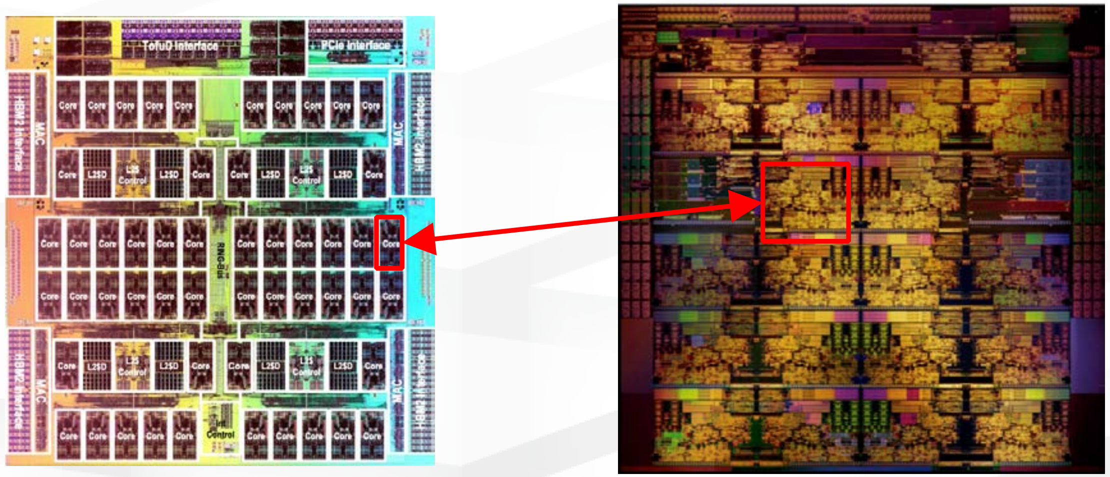
- **HPC-oriented design**
  - Small core ⇒ Less O3 resources
  - (Relatively) Long pipeline
    - 9 cycles for floating point operations
    - Core has only L1 cache
  - High-throughput, but long-latency
  - Pipeline often stalls
    for loops having complex body.

	A64FX	Skylake
ReOrder Buffer	128 entries	224 entries
Reservation Station	60 (=10x2+20x2) entries	97 entries
Physical Vector Register	128 (=32 + 96) entries	168 entries
Load Buffer	40 entries	72 entries
Store Buffer	24 entries	56 entries

A64FX : https://github.com/fujitsu/A64FX
Skylake : https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(server)

- A64FX: 52 cores (48 cores), 400 mm² die size (8.3 mm²/core), 7nm FinFET process (TSMC)
- Xeon Skylake: 20 tiles (5x4), 18 cores, ~485 mm² die size (estimated) (26.9 mm²/core), 14 nm process (Intel)
- A64FX core is more than 3 times smaller per core.

A64FX:
400 mm²
(20 x 20)



https://www.fujitsu.com/jp/solutions/business-technology/tc/catalog/ff2019-post-k-computer-development.pdf

https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(server)

Xeon Skylake, High Core Count:
4 x 5 tiles, 18 cores, 2 tiles used for memory interface
485 mm² (22 x 22)

Performance Tuning for A64FX processor

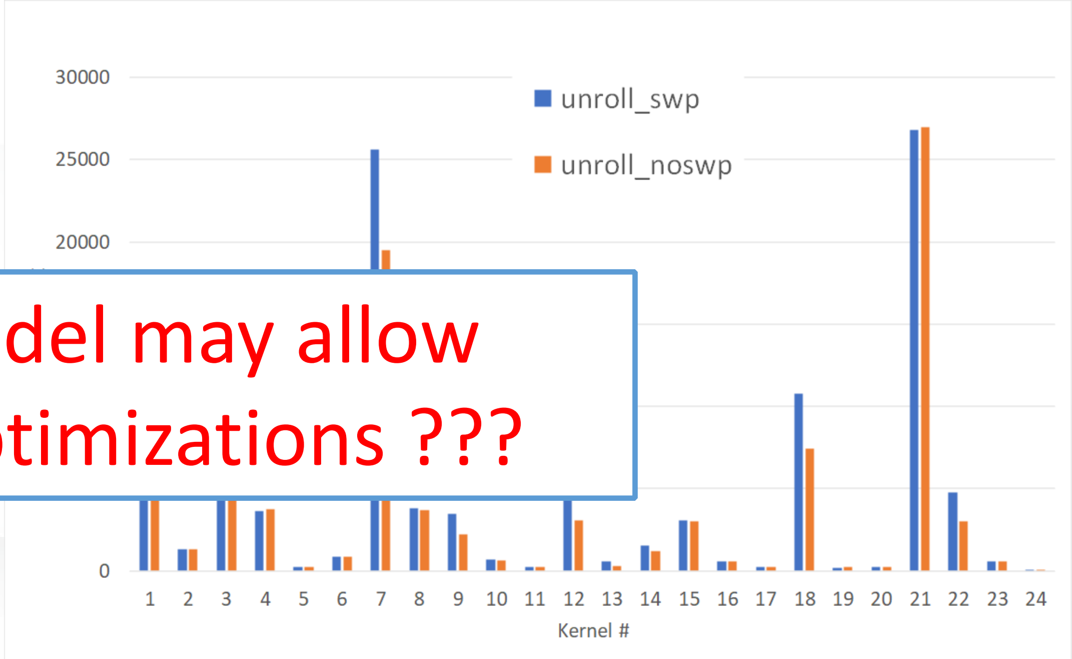
- **HPC-oriented design**
  - Small core ⇒ Less O3 resources
  - (Relatively) Long pipeline
    - 9 cycles for floating point operations
    - Core has only L1 cache
  - High-throughput, but long-latency
  - Pipeline often stalls
    for loops having complex body.

	A64FX	Skylake
ReOrder Buffer	128 entries	224 entries
Reservation Station	60 (=10x2+20x2) entries	97 entries
Physical Vector Register	128 (=32 + 96) entries	168 entries
Load Buffer	40 entries	72 entries
Store Buffer	24 entries	56 entries

A64FX : https://github.com/fujitsu/A64FX
Skylake : https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(server)

- **Compiler optimization (Fujitsu compiler)**
  - SWP: software pipelining
    - ～ 20% spe
  - Automatic ar

Kernel programming model may allow more aggressive loop optimizations ???



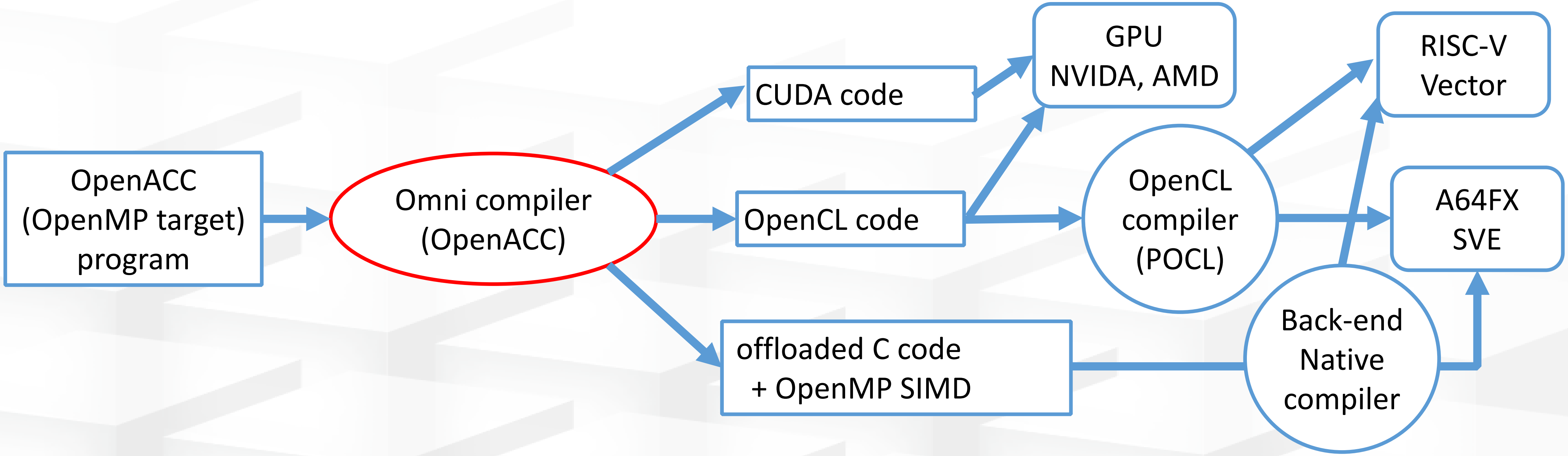
Performance improvement by SWP in Livermore Kernels by Fujitsu compiler

Motivation and Objectives of OpenACC for A64FX

- **OpenACC as a Programming model to exploit parallelism of A64FX architecture**
  - HPC oriented design of 64FX
  - GPU-like code generation may improve the performance?
- **OpenACC as a Programming model for many core wide-SIMD processors**
  - Not only 64FX. Recent AMD/Intel high-end processors have many cores (more than 64) with SIMD
  - OpenACC may provide other models to describe parallelism of NUMA/core/vector more than “classic” OpenMP with sperate memory space management.
- **Porting OpenACC apps to A64FX and Fugaku**
  - Many apps are already written in OpenACC

Our compiler infrastructure: Omni Compiler

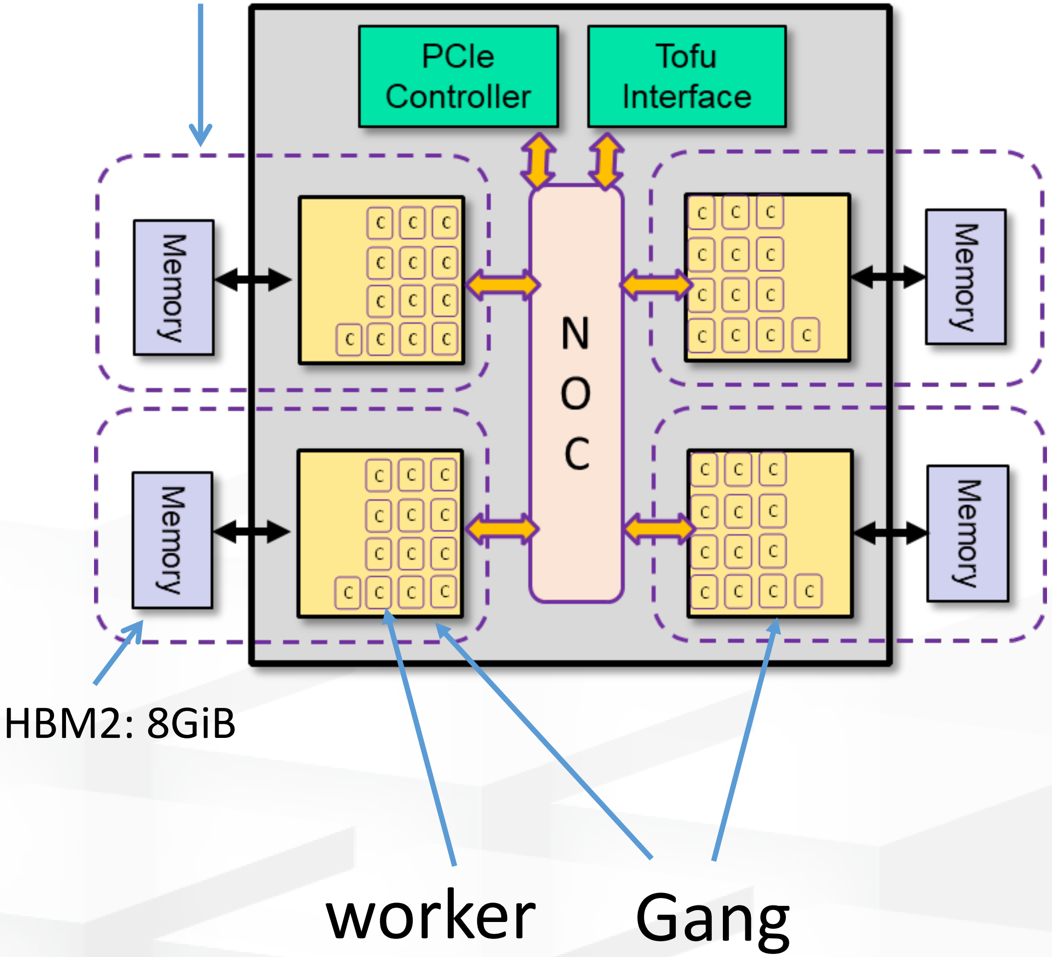
- **Omni compiler is a source-to-source compiler infrastructure**
  - Omni OpenACC compiler for CUDA is already available
- **Currently, we are working on the following two approaches:**
  - Generating OpenCL code, which is to be compiled by POCL for A64FX
  - Generating translated C code with OpenMP SIMD directive for vector



OpenACC execution model for A64FX

- **OpenACC supports 3-level parallelism**
  - Gang = PEs
    - SM in case of NVIDIA GPU
    - CMG (NUMA node) in case of A64FX
  - Worker = each PE
    - Wraps in case of NVIDIA GPU
    - Core in case of A64FX
  - Vector = each threads in PE
    - Threads in a warp in case of NVIDIA GPU
    - SIMD in case of A64FX

CMG(Core-Memory-Group): NUMA node
12+1 core



Execution Environment of offloaded code

- **Execution by pthread**
  - Threads are used to execute offloaded code
  - Allocate sperate memory for offloaded code and share the memory space with "host" code
  - Easy to move and share the data, but may have side effect from "host"

- **Execution by process**
  - Invoke other process to execute offloaded code
  - Move the data by IPC mechanism such as "mmap" system calls
  - The process has the different OS resources such as TLBs so that it may reduce "side effect".
  - Overhead of process switching

Example of translation

- Translation to C code with OpenMP SIMD directive for vectorization
- Execution Environment by pthread
- Example: non-blocking matrix-matrix multiply

Matrix-matrix multiply by inner product

```
#define MAT(a,n_size,i,j)  a[n_size*(i)+(j)]
…
#pragma acc parallel
{
#pragma acc loop gang
for (int i = 0; i < n; i++)
 for(int j = 0; j < n; j++){
   double ip = 0.0;
#pragma acc loop vector
   for(int k = 0; k < n; k++)
     ip += MAT(x1,n,i,k)*MAT(x2,n,k,j);
   MAT(y,n,i,j) = ip;
  }
}
```

```
// executed by thread running
//      on each core
… Schedule iterations …
… assigned iterations given by b and e …
…
for (int i = b; i < e; i++)
 for(int j = 0; j < n; j++){
   double ip = 0.0;
#pragma omp simd
   for(int k = 0; k < n; k++)
     ip += MAT(x1,n,i,k)*MAT(x2,n,k,j);
   MAT(y,n,i,j) = ip;
  }
```

Example of translation

- Translation to C code with OpenMP SIMD directive for vectorization
- Execution Environment by pthread
- Example: non-blocking matrix-matrix multiply

Matrix-matrix multiply by cross product

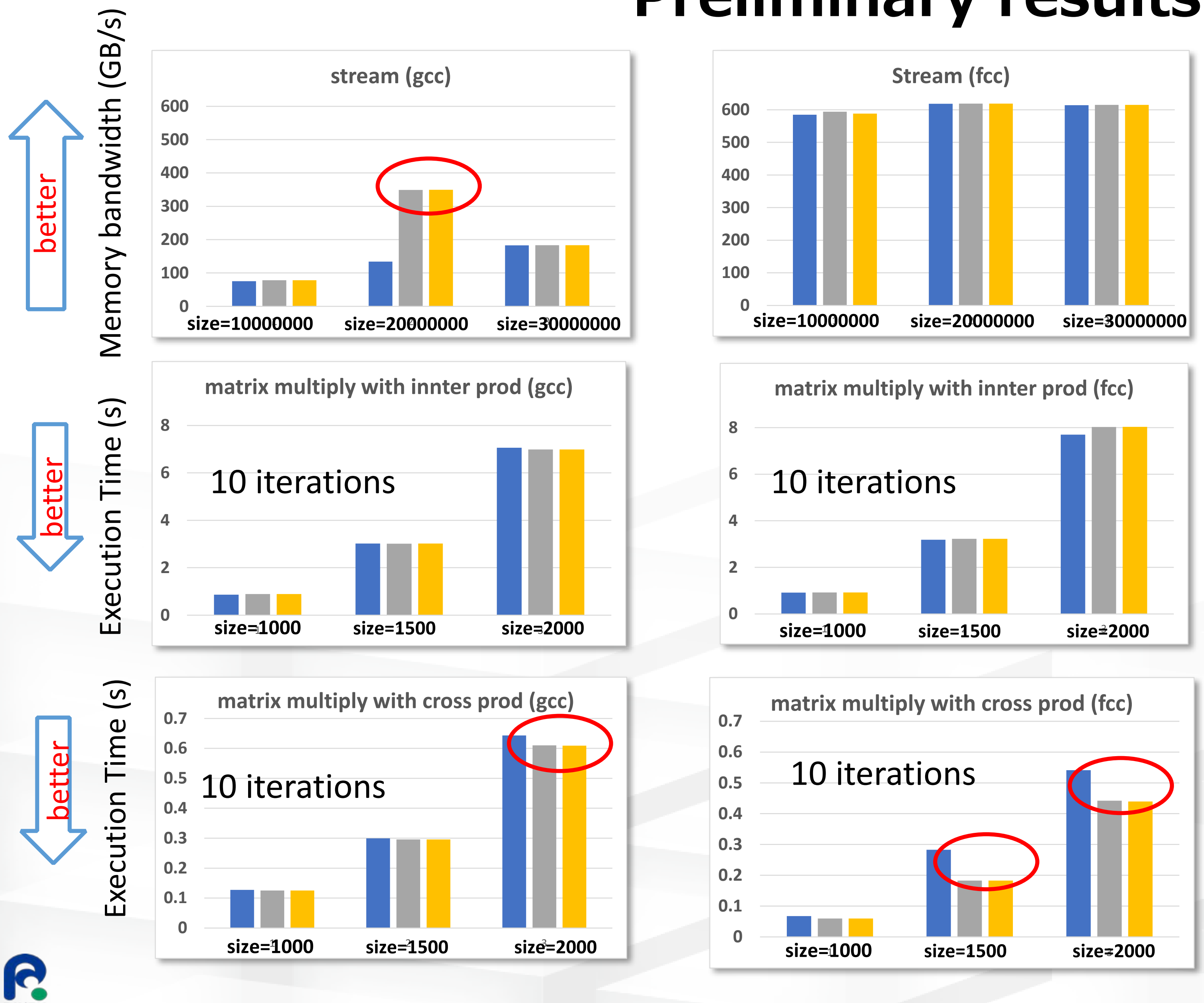
```
#define MAT(a,n_size,i,j)  a[n_size*(i)+(j)]
…
#pragma acc parallel
{
#pragma acc loop
for (int i = 0; i < n; i++)
      for(int j = 0; j < n; j++)
        MAT(y,n,i,j) = 0.0;
#pragma acc loop gang
for (int i = 0; i < n; i++)
   for(int k = 0; k < n; k++){
     double c = MAT(x1,n,i,k);
#pragma acc loop vector
     for(int j = 0; j < n; j++)
       MAT(y,n,i,j) += c*MAT(x2,n,k,j);
   }
```

```
// executed by thread running
//      on each core
… Schedule iterations …
… assigned iterations given by b and e …
…
for (int i = b; i < e; i++)
   for(int k = 0; k < n; k++){
     double c = MAT(x1,n,i,k);
#pragma omp simd
     for(int j = 0; j < n; j++)
       MAT(y,n,i,j) += c*MAT(x2,n,k,j);
   }
```

Preliminary evaluation

- **Translation to C code with OpenMP SIMD directive for vectorization**
- **Benchmark**
  - Stream benchmark
  - Matrix-multiplication by inner product
  - Matrix-multiplication by cross product
- **Backend compilers**
  - Fujitsu
  - gcc
  - llvm

Preliminary results



🟦 OpenMP

⬜ Translated C coded from OpenACC

🟨 Translated C code with SIMD directive from OpenACC

- **Backend compilers**
  - gcc 11.2
    - -fopenmp -march=armv8.2-a+sve -msve-vector-bits=512 -mtune=a64fx -ffast-math -O3
  - Fujitsu compiler (fcc)
    - -Kfast –fopenmp
- **Some improvements were found because the transformation helps good vectorization for inner loop**
- **In most cases, the performance is the same as OpenMP**
- **No performance improvement by SIMD directive since vectorization is done by compiler.**
- **Inner production version is very slower than cross product version.**

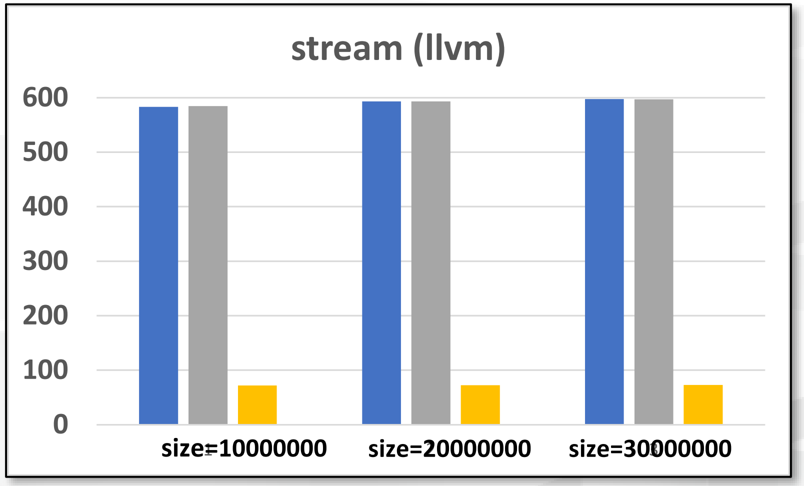
Preliminary results

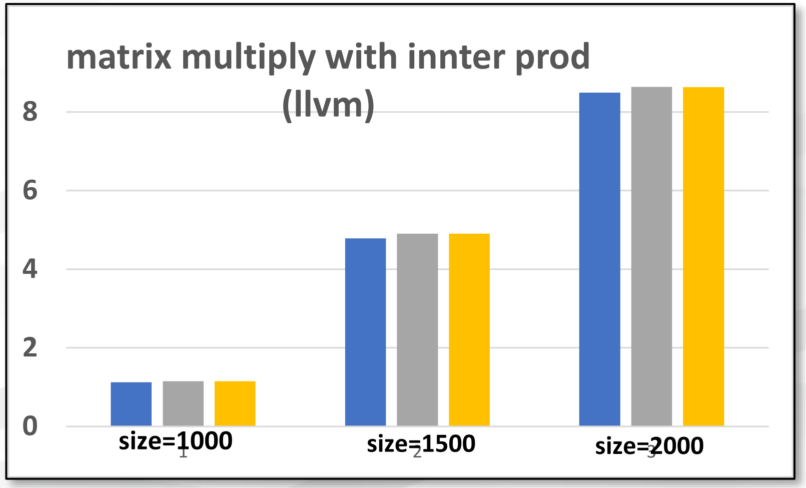
- **Backend compiler**
  - llvm 12.0.1
    - -fopenmp -Ofast -ffast-math -march=armv8-a+sve -ffp-contract=on -mllvm -aarch64-sve-vector-bits-min=512 –mllvm -aarch64-sve-vector-bits-max=512
  - Some improvements were found as fcc

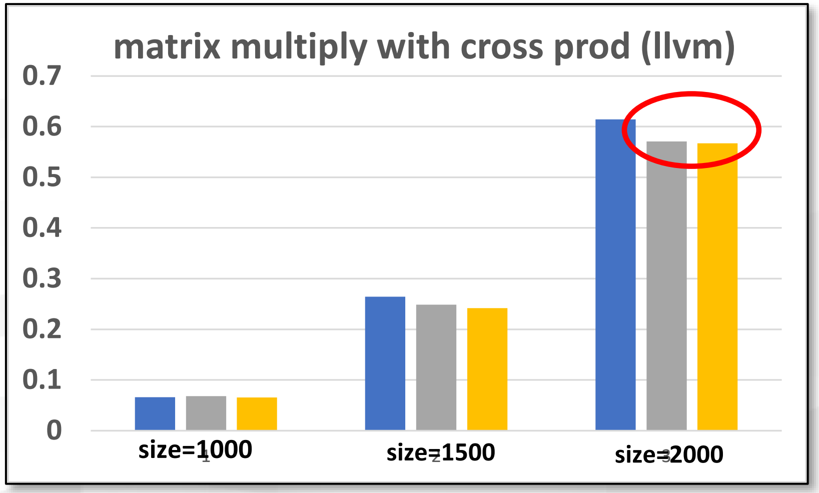
🟦 OpenMP

◼ Translated C coded from OpenACC

🟨 Translated C code with SIMD directive from OpenACC

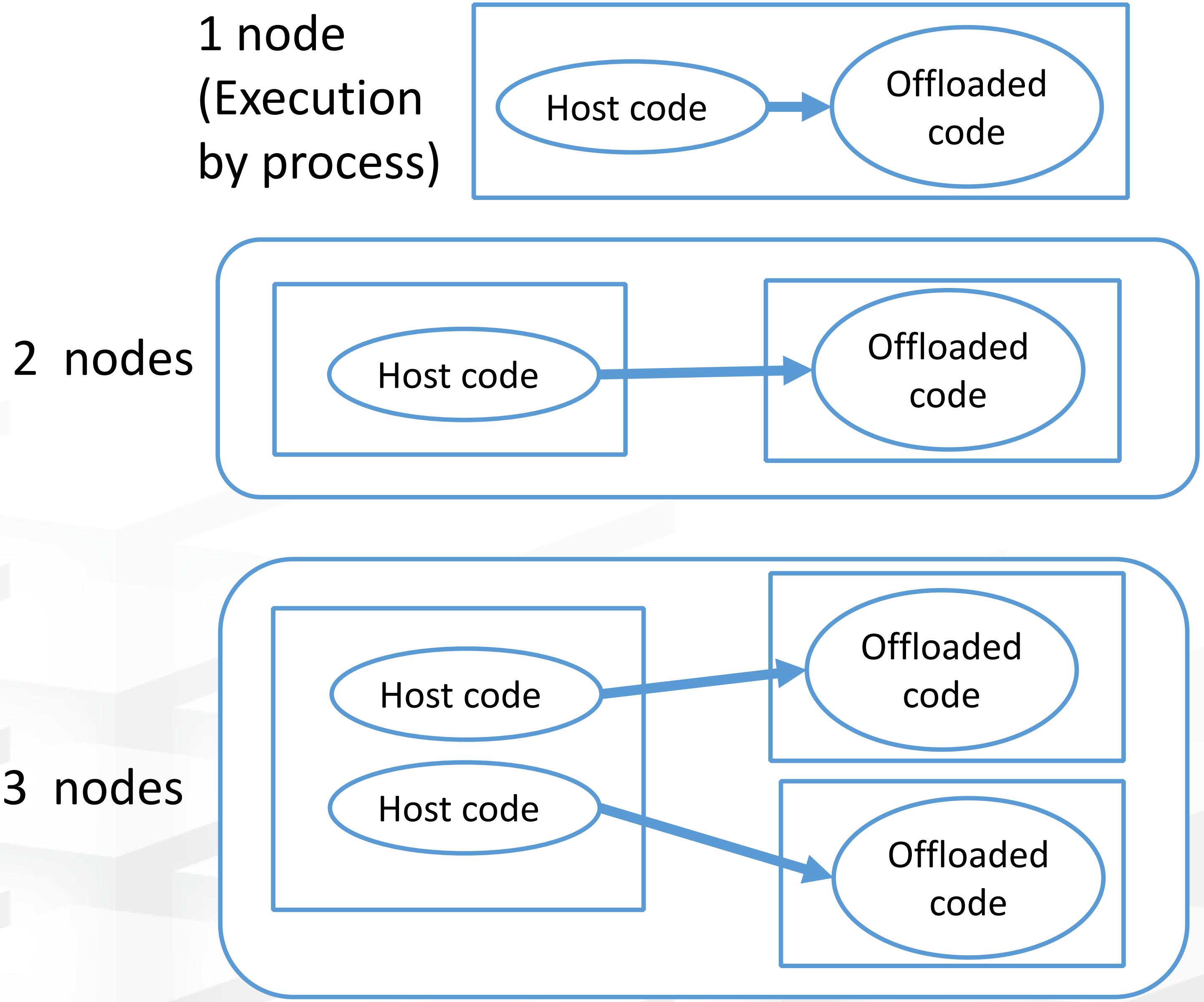






Offload to other nodes

- **“Execution by process” can be extended to “execution by other nodes”**
  - Useful to exploit other parallelism
  - Need asynchronous offloading
- **Question: may it be better than all nodes used as MPI process?**



1 node (Execution by process)

Host code → Offloaded code

2 nodes

Host code → Offloaded code

3 nodes

Host code → Offloaded code

Host code → Offloaded code

Concluding remarks

- **We are currently working on OpenACC compiler for A64FX**
  - Several design choices such as code generation and execution environments.
  - We expect that OpenACC programming model provides other possibilities to exploit parallelism of manycore with wide-SIMD using NUMA/core/vectors
  - It may be applied to recent high-end processors such as AMD and Intel.
- **Our preliminary experimental results still need to be examined.**
  - We need to examine more complicated loops by OpenACC, and other implementation of execution environment such as POCL and by process
- **We are also working on OpenCL (POCL) which can be used as a backend of our OpenACC compiler**
  - OpenCL kernel programming is expected to allow an aggressive optimization.
- **Evaluation and tuning of OpenACC gcc (or llvm?) as a native compiler for A64FX**
  - The current version of "gcc –fopenacc" for A64FX has a problem (not executed in parallel?)