

Programming Environment for FugakuNEXT

Hitoshi Murai, Lead of Programming Env. WG, RIKEN R-CCS

Introduction

- **A FugakuNEXT compute node will consist of:**
 - **Fujitsu MONAKA-X CPUs, and**
 - **NVIDIA GPUs.**
- **Programming models supporting both two are needed for users to achieve good performance on FugakuNEXT.**
- **In particular, those for programming GPUs will be essential.**

Agenda

- **Possible programming models for FugakuNEXT**
- **Comparison**
- **Basic policy on programming model for FugakuNEXT**
- **Additional related technologies**
- **Summary**

Programming Models for FugakuNEXT

- **Directive-based**
 - OpenACC
 - OpenMP
- **CUDA**
- **Standard Language Parallelism**
 - `stdpar` for C++
 - `DO CONCURRENT` for Fortran
- **Portable programming models**
 - Kokkos
 - (SYCL)

OpenACC

- Directive-based model for heterogeneous computing, especially GPUs.
- Introduced in 2011 by Cray, CAPS, NVIDIA, and PGI.
 - currently supported only by NVIDIA.
- Supported Languages: C, C++, Fortran



```
#pragma acc parallel loop
for (int i = 0; i < N; i++) {
    A[i] = B[i] + C[i];
}
```

OpenMP



- Directive-based API for shared-memory parallel programming.
- First specification in 1997 by a consortium of HPC vendors.
- Accelerator offloading added from V4.0 in 2023.
- Supported Languages: C, C++, Fortran
- Supported by every compilers.
- Offloading in OpenMP often performs worse than OpenACC because compilers are not yet mature.

```
#pragma omp target parallel for ¥  
    map(to: B[0:N], C[0:N]) ¥  
    map(from: A[0:N])  
for (int i = 0; i < N; i++) {  
    A[i] = B[i] + C[i];  
}
```

CUDA

- NVIDIA's parallel computing platform and API for programming GPUs directly.
- Released in 2007 by NVIDIA, enabling general-purpose GPU computing (GPGPU).
- Supported Languages: C, C++, Fortran, Python (via PyCUDA, Numba), others via bindings



```
__global__ void add(int *A, int *B, int *C, int N) {  
    int i = blockIdx.x * blockDim.x + threadIdx.x;  
    if (i < N) C[i] = A[i] + B[i];  
}
```

Standard Language Parallelism

- **Standard syntax to assert parallelism of a loop.**
 - `stdpar` **supported since C++17.**
 - `DO CONCURRENT` **supported since Fortran 2008.**
- **Compilers *can* parallelize/offload the loop for themselves.**

```
std::transform(std::execution::par,  
              B.begin(), B.end(),  
              C.begin(), A.begin(),  
              [](int b, int c) { return b + c; });
```

- **might not perform well because they assert no additional information other than “parallelism.”**

```
do concurrent (i = 1:N)  
  A(i) = B(i) + C(i)  
end do
```

Kokkos

- An open-source C++ library for performance portability across CPUs, GPUs, and accelerators.
- Developed by Sandia National Laboratories starting around 2012 in the Exascale Computing Project (ECP); recently adopted by CEA in France.
 - Many applications developed in ECP are implemented with Kokkos.
 - Dedicated tools such as a profiler and debugger and mathematical libraries (i.e. Kokkos Kernels) are available.
- The portability depends on available multiple backends such as OpenMP, CUDA, SYCL, and HIP.
- Now one of the projects of High Performance Software Foundation (HPSF).



```
Kokkos::parallel_for(N,  
  KOKKOS_LAMBDA(const int i) {  
    A(i) = B(i) + C(i);  
  });
```

Comparison

	Performance	Easiness	Portability	Note
OpenACC	△	○	△	In effect, only supported by NVIDIA
OpenMP	×	○	○	Performance should improve if compilers mature.
CUDA	○	×	×	
Standard Language Parallelism	×	○○	○	
Kokkos	△ (depends on backends)	△	○	

Basic Policy on Programming Model for FugakuNEXT

- OpenACC is a central means for offloading.
- Recommend Kokkos
 - for brand-new C/C++ projects; and
 - (possibly) for brand-new Fortran projects because a tool for Kokkos-Fortran interoperability will be provided.
- It can be expected that the NVIDIA/Fujitsu compilers support OpenMP, CUDA, and standard language parallelism for themselves.
- Others (e.g. SYCL) should also be available as external packages.

Basic Design of Kokkos-Fortran Interop. Tool

- **A users does:**
 - **insert a directive before the target loop (in `sub.f90`); and**
 - **add initialization/finalization in the main program.**

- **The tool does:**
 - A) translate `sub.f90`;**
 - B) generate `kokkos_sub.cpp` (a Kokkos code); and**
 - C) generate `sub_mod.f90` (a module to define interfaces).**

```

subroutine sub
real :: x(xil:xiu,xjl:xju), y(yil:yiou,yjl:yju), a      sub.f90
!$kks parallel_for data(x,y,a) on(i,j) tile(t1,t2)
do j=jl, ju
  do i=il, iu
    y(i,j) = y(i,j) + a * x(i,j)
  end do
end do
end

```



automatically translated
or generated by the tool

```

subroutine sub                                     sub.f90
use :: sub_kokkos_mod
real :: x(xil:xiu,xjl:xju), y(yil:yiou,yjl:yju), a
call kokkos_sub0(to_nd_array(x), to_nd_array(y), a, ! data
                xil, xjl, yil, yjl,                ! lb of arrays
                il, iu, jl, ju,                    ! lb and ub of loops
                t1, t2)                            ! tiling params.
end

```



```

#include <Kokkos_Core.hpp>
#include "flcl-cxx.hpp"

extern "C" {
  void kokkos_sub0(flcl_ndarray_t *nd_array_x,
                  flcl_ndarray_t *nd_array_y,
                  float a,
                  int xil, int xjl, int yil, int yjl,
                  int il, in iu, int jl, int ju,
                  int t1, int t2){

    auto x = flcl::view_from_ndarray<float**>(*nd_array_x);
    auto y = flcl::view_from_ndarray<float**>(*nd_array_y);

    parallel_for("sub0", Kokkos::MDRangePolicy<Kokkos::Rank<2>>({il,jl},{iu+1,ju+1},{t1,t2}),
                KOKKOS_LAMBDA(int i, int j) {
                  y(i-yil,j-yjl) = y(i-yil,j-yjl) + a * x(i-xil,j-xjl);
                });
  }
}

```



```
module sub_kokkos_mod

use, intrinsic :: iso_c_binding
use :: flcl_mod

interface
  subroutine kokkos_sub0(nd_array_x, nd_array_y, a, xil, xjl, yil, yjl, il, iu, jl, ju, t1, t2) bind (c)
    use, intrinsic :: iso_c_binding
    use :: flcl_mod
    type(nd_array_t) :: nd_array_x
    type(nd_array_t) :: nd_array_y
    real(c_float) :: a
    integer(c_int) :: xil, xjl, yil, yjl
    integer(c_int) :: il, iu, jl, ju
    integer(c_int) :: t1, t2
  end subroutine
end interface

end module
```

Conversion between OpenMP and OpenACC

- Some tools support the conversion from OpenMP to OpenACC and/or vice versa.
 - *Clacc* translates OpenACC to OpenMP (ORNL)
 - Intel Application Migration Tool for OpenACC to OpenMP API for C/C++/Fortran (Intel)
 - *ACC2OMP* for Fortran (UIUC ?)
- SOLOMON (U. Tokyo)
 - a *unified* directive to be substituted to OpenACC or OpenMP.

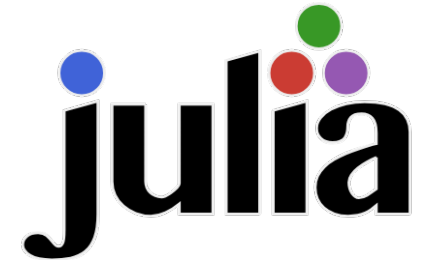
```
OFFLOAD(AS_INDEPENDENT, NUM_THREADS(NTHREADS))
for (int32_t i= 0; i<N; i++) {
```

```
#pragma acc kernels vector_length(NTHREADS)
#pragma acc loop independent
for (int32_t i= 0; i<N; i++) {
```

```
#pragma omp target teams loop thread_limit(NTHREADS)
for (int32_t i= 0; i<N; i++) {
```

JACC (Julia Accelerated)

- A Julia library for a unified front-end on top of multiple backends (just like Kokkos).
- Julia is a high-level, high-performance programming language designed for numerical computing, data science, and scientific applications.
- First released in 2012; official 1.0 release in 2018.
- Dynamically typed, JIT-compiled via LLVM.
- Combines ease of Python/R with speed close to C/Fortran.
- Rich ecosystem for machine learning, scientific computing, and parallelism.



```
for i in 1:N
    A[i] = B[i] + C[i]
end
```

Summary

- For programming models on FugakuNEXT,
 - OpenACC is a central means for offloading;
 - Kokkos is also recommended particularly for brand-new C/C++ projects; and
 - we are developing a tool for Kokkos-Fortran interoperability.
- We are studying additional technologies including:
 - conversion between OpenMP and OpenACC
 - JACC