OAK RIDGE National Laboratory

Performance in Julia (Matrix-Matrix Multiplication)

Pedro Valero-Lara and William Godoy

Computer Scientist at Computer Science and Mathematics Division, Programming Systems Group

valerolarap@ornl.gov

ORNL is managed by UT-Battelle LLC for the US Department of Energy

EXASCALE COMPUTING PROJECT

U.S. DEPARTMENT OF ENERGY

Motivation

- What is the position of Julia in HPC?
  - We use a very simple 3 for-loop nested matrix-matrix multiplication https://github.com/williamfgc/simple-gemm
  - Kokkos, OpenMP, Julia, Python+Numba
  - Exascale architectures (Crusher/Frontier)
    - CPU -> 64-core AMD EPYC 7A53
    - GPU -> AMD MI250X

Which programming language/model is each one?

https://www.menti.com/alhjr2qoabtk (4459 1234)

```cpp
int64_t i, k, j;
float temp;
#pragma omp parallel for
  default(shared) private(i, k, j, temp)
for (i = 0; i < A_rows; i++) {
  for (k = 0; k < A_cols; k++) {
    temp = A[i * A_cols + k];
    for (j = 0; j < B_cols; j++) {
      C[i * B_cols + j] += temp * B[k * B_cols + j];
    }
  }
}
```

1

```fortran
integer(kind=8) :: i, l, j
real(kind=4) :: temp
!$omp parallel do default(shared) private(j,l,i, temp)
do j=1,B_cols
  do l=1,A_cols
    temp = B(l,j)
    do i=1,A_rows
      C(i,j) = C(i,j) + temp * A(i,l)
    end do
  end do
end do
!$omp end parallel do
```

2

```cpp
int row = blockIdx.y * blockDim.y + threadIdx.y;
int col = blockIdx.x * blockDim.x + threadIdx.x;
double sum = 0.0;
if( col < k && row < m)
{
  for(int i = 0; i < n; i++) {
    sum += a[row * n + i] * b[i * k + col];
  }
  c[row * k + col] = sum;
}
```

3

```cpp
Kokkos::parallel_for( "AxB-C",
  mdrange_policy( {0, 0}, {M, N}),
  KOKKOS_LAMBDA ( int m, int n ){
    float tmp = 0.0;
    for ( int k = 0; k < K; k++ ) {
      tmp += A(m, k) * B(k, n);
    }
    C(m, n) = tmp;
  }
);
```

4

```julia
@njit(parallel=True, nogil=True, fastmath=True)
...
for i in prange(0, A_rows):
  for k in range(0, A_cols):
    temp = A[i, k]
    for j in range(0, B_cols):
      C[i, j] += temp * B[k, j]
```

5

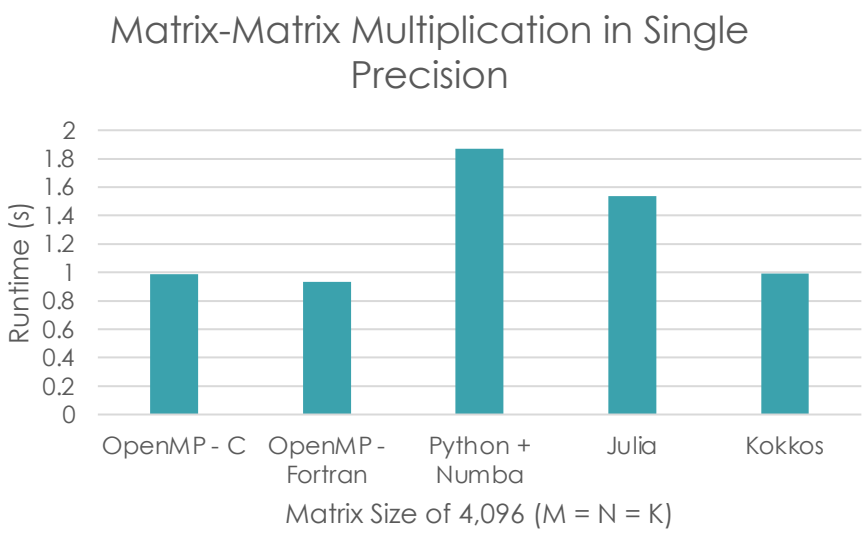
```julia
Threads.@threads for j = 1:B_cols
  for l = 1:A_cols
    @inbounds temp::Float32 = B[l, j]::Float32
    for i = 1:A_rows
      @inbounds C[i, j] += temp * A[i, l]
    end
  end
end
```

6

```julia
row = (AMDGPU.workgroupIdx().x - 1) *
       AMDGPU.workgroupDim().x + AMDGPU.workitemIdx().x
col = (AMDGPU.workgroupIdx().y - 1) *
      AMDGPU.workgroupDim().y + AMDGPU.workitemIdx().y
sum = Float32(0.0)
if row <= size(A, 1) && col <= size(B, 2)
  for i = 1:size(A, 2)
    @inbounds sum += A[row, i] * B[i, col]
  end
  C[row, col] = sum
end
```

7

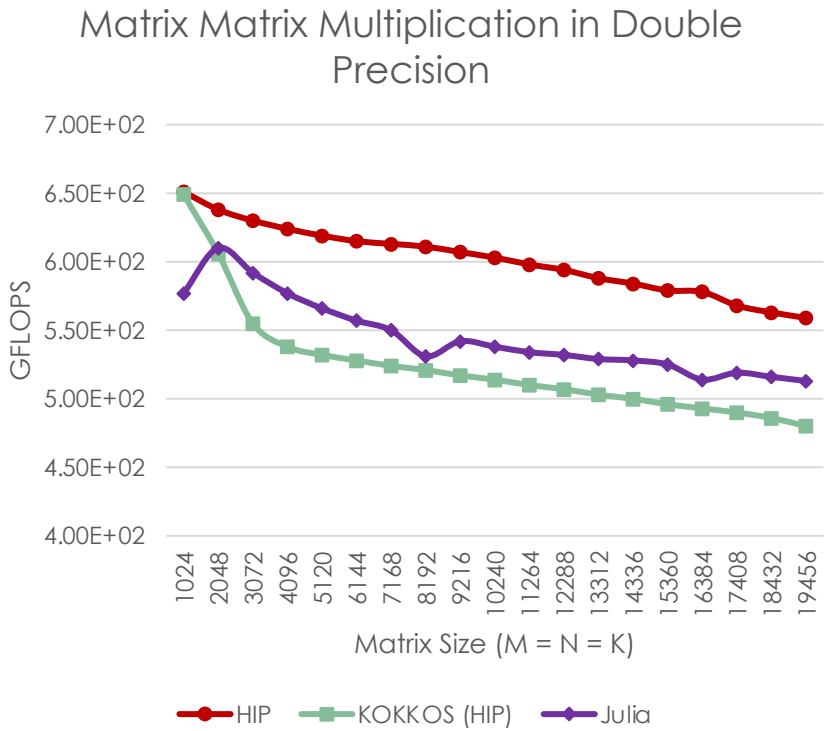
Performance Numbers on 1x AMD CPU EPYC 7A53 64-Core (Frontier)



https://github.com/williamfgc/simple-gemm

- **OpenMP:**
  - OMP_NUM_THREADS=64, OMP_PROC_BIND=spread, OMP_PLACES=threads
  - Modules loaded: rocm/5.2.0 or PrgEnv-gnu/8.3.3
  - **C:**
    - Compilers: AMD clang version 14.0.0 (amdclang), -march=znver3 -mtune=znver3 -fopenmp=libomp –O3
      (GCC) 11.2.0 20210728 (Cray Inc.) (gcc), -fopenmp -O3
  - **Fortran:**
    - Compilers: AMD flang-new version 14.0.0 (amdflang), -march=znver3 -mtune=znver3 –fopenmp –O3
      GNU Fortran (SUSE Linux) 7.5.0 (gfortran), -fopenmp -O3
  - Similar performance using AMD clang/flang and GNU GCC/Fortran
- **Kokkos (OpenMP):**
  - Kokkos 3.5.0, KOKKOS_DEVICES=OpenMP, (AMD clang) KOKKOS_ARCH = "Zen3",
    (GCC) KOKKOS_ARCH="BSD"
  - OMP_NUM_THREADS=64, OMP_PROC_BIND=spread, OMP_PLACES=threads
  - Modules loaded: rocm/5.2.0 or PrgEnv-gnu/8.3.3
  - Compilers: AMD clang version 14.0.0 (amdclang++), -std=c++14 -march=znver3 -mtune=znver3 -fopenmp=libomp –O3
    (GCC) 11.2.0 20210728 (Cray Inc.) (g++), -std=c++14 -fopenmp -O3
  - Similar performance using AMD clang and GNU GCC
- **Python+NUMBA:**
  - Python 3.9.12 and NUMBA 0.55.2
    EXECUTABLE=../../python/GemmDenseThreads.py
    python3 -64 --project=$GemmDenseDIR $EXECUTABLE $M $M $M
- **Julia:**
  - julia 1.8.0-rc1:
    https://julialang.org/downloads/#upcoming_release
    PROJDIR=../../julia/GemmDenseThreads
    EXECUTABLE=$PROJDIR/gemm-dense-threads.jl
    srun -n 1 --ntasks-per-node=1 -c julia -O3 -64 --project=$PROJDIR $EXECUTABLE $M $M $M

Performance Numbers on 1x AMD GPU MI250x (Frontier)



https://github.com/williamfgc/simple-gemm

- **HIP:**
  - Modules loaded: rocm/5.2.0
  - Compiler: HIP version: 5.2.21151-afdc89f8 AMD clang version 14.0.0
- **Kokkos (HIP):**
  - Kokkos 3.5.0 (installed in Crusher)
    hipcc -std=c++14
    -I/sw/crusher/spack-envs/base/opt/cray-sles15-zen3/cce-14.0.0/kokkos-3.5.00-wnl6ceqkrlte4qbopksgwnyfwk4arjyo/include
    -L/sw/crusher/spack-envs/base/opt/cray-sles15-zen3/cce-14.0.0/kokkos-3.5.00-wnl6ceqkrlte4qbopksgwnyfwk4arjyo/lib64
    -lkokkoscore test.cpp –o run
    export LD_LIBRARY_PATH=
    $LD_LIBRARY_PATH:/sw/crusher/spack-envs/base/opt/cray-sles15-zen3/cce-14.0.0/kokkos-3.5.00-wnl6ceqkrlte4qbopksgwnyfwk4arjyo/lib64
- **Julia:**
  - julia 1.8.0-rc1:
    https://julialang.org/downloads/#upcoming_release
  - Run on Crusher:
    export JULIA-AMDGPU_DISABLE_ARTIFACTS=1
    https://github.com/williamfgc/simple-gemm/blob/main/scripts/julia/run_amdgpu_crusher.sh
- **Python+Numba:**
  - Note that the support of NUMBA for AMD GPUs is “unmaintained”:
    https://numba.readthedocs.io/en/stable/release-notes.html?highlight=ROCm#version-0-54-0-19-august-2021

Performance Portability (Julia vs Python+Numba vs Kokkos)

- **Performance Portability (ΦM) [1]**

$$\Phi_M = \frac{\sum_{i \in T} e_i(a)}{|T|}$$

  - e_i (a) -> efficiency of one portable code (i) on an application (matrix-matrix multiplication)
  - T -> number of platforms

- **Efficiency**
  - Time consumed by the portable model divided by the time consumed by the no-portable and specific model

$$e_{MI250x} = \frac{Julia\ Performance}{HIP\ Performance}$$

- **Architectures tested**
  - 2 CPUs architectures
    - ARM Wombat (ARM) Ampere Altra Ampere Altra
    - AMD Epyc 7A53 x 64-core, 4-NUMA
  - 2 GPUs
    - Wombat (NVIDIA) Crusher (AMD)
    - Model A100 Ampere MI250X

Architecture	Kokkos	Julia	Python/Numba
Double precision			
$e_{Epyc\ 7A53}$	**0.994**	0.912	0.550
$e_{Ampere\ Altra}$	0.854	**0.907**	0.713
e_{MI250x}	0.842	**0.903**	-
e_{A100}	0.260	**0.867**	0.130
Single precision			
$e_{Epyc\ 7A53}$	**1.014**	0.976	0.655
$e_{Ampere\ Altra}$	0.836	**0.900**	0.400
e_{MI250x}	0.677	**1.050**	-
e_{A100}	0.208	**0.600**	0.095

https://github.com/williamfgc/simple-gemm

	Kokkos	Julia	Python/Numba
Double precision			
Φ_M	0.738	**0.897**	0.348
Single precision			
Φ_M	0.684	**0.882**	0.288

[1] A. Marowka, "On the performance portability of openacc, openmp, kokkos and RAJA," in HPC Asia 2022: International Conference on High Performance Computing in Asia-Pacific Region, Virtual Event, Japan, January 12 - 14, 2022. ACM, 2022, pp. 103–114. [Online]. Available: https://doi.org/10.1145/3492805.3492806

OAK RIDGE National Laboratory

Thanks! Questions??

Pedro Valero-Lara and William Godoy

Computer Scientist at Computer Science and Mathematics Division, Programming Systems Group

valerolarap@ornl.gov

ORNL is managed by UT-Battelle LLC for the US Department of Energy

EXASCALE COMPUTING PROJECT

U.S. DEPARTMENT OF ENERGY