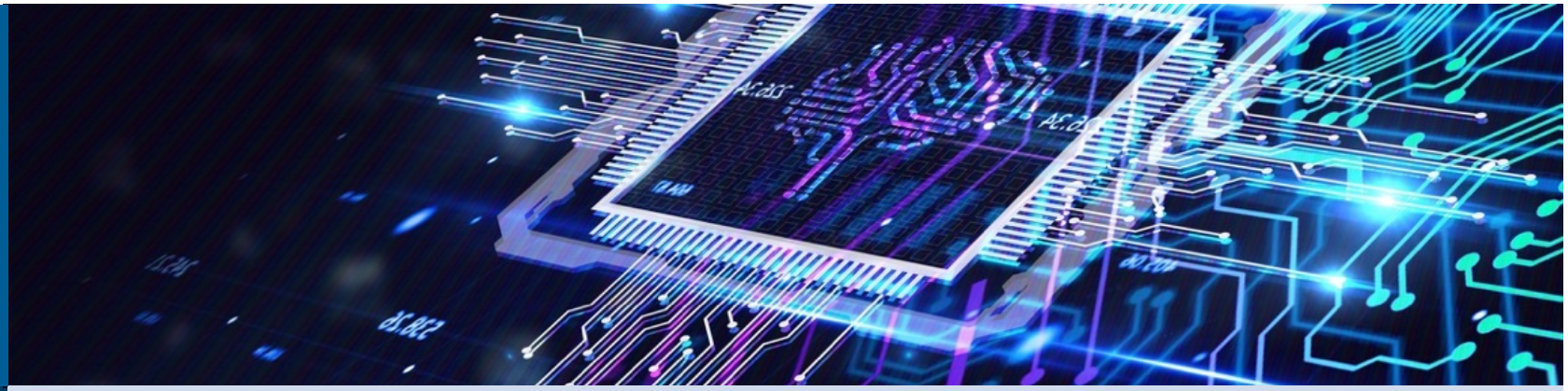
CSC

ICT Solutions for Brilliant Minds



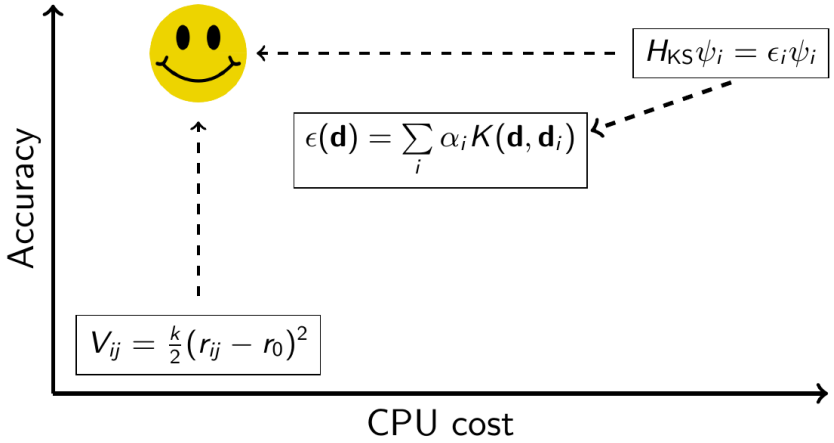
TurboGAP Porting to GPUs

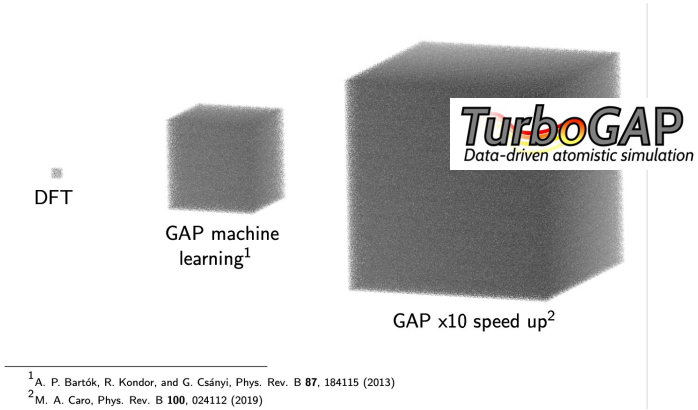
Cristian-Vasile Achim, Martti Luohivuori, Miguel Caro, Jussi Heikonen

Cray User Group Meeting – May 7 –11 , 2023

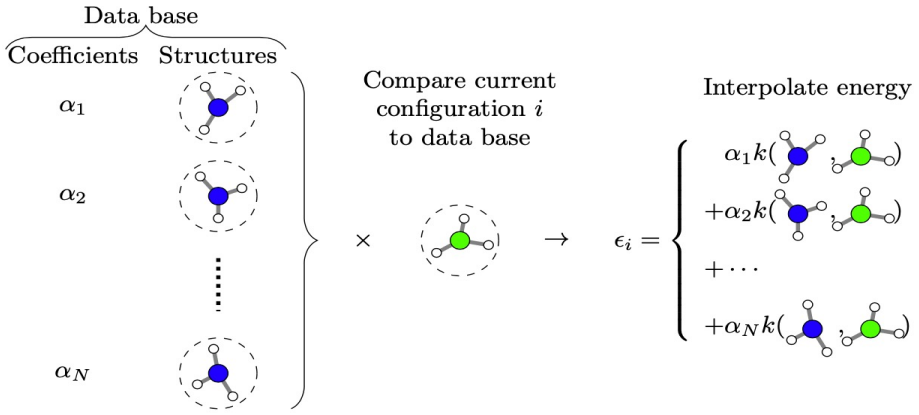
Background

TurboGAP (MD with Kernel Based Machine Learning model)





[1] A. P. Bartók, R. Kondor, and G. Csányi, Phys. Rev. B **87**, 184115 (2013)
[2] M. A. Caro, Phys. Rev. B **100**, 024112 (2019)



$$\bar{\epsilon}(\mathbf{q}_i) = \delta^2 \sum_{s \in S} \alpha_s k(\mathbf{q}_i, \mathbf{q}_s)$$

$$\frac{\partial \bar{\epsilon}(\mathbf{q}_i)}{\partial r_{a,k}} = \delta^2 \sum_{s \in S} \alpha_s \nabla_{\mathbf{q}_i} k(\mathbf{q}_i, \mathbf{q}_s) \cdot \frac{\partial \mathbf{q}_i}{\partial r_{a,k}}$$

Applications

Present and Near Future:

- Potentials for various materials: Pt, Au, Fe, Si, Cu, CuAu, PtAu
- Amorphous carbon, C_{60} , nanotube, nano-porous (for battery applications)
- Pyrolysis of biomass
- Copper based materials with focused efficient conversion of CO_2 to methanol

Long Term Aim:

- Biological processes: Protein folding, Lipid membrane, Drug discovery

Scalability Tests

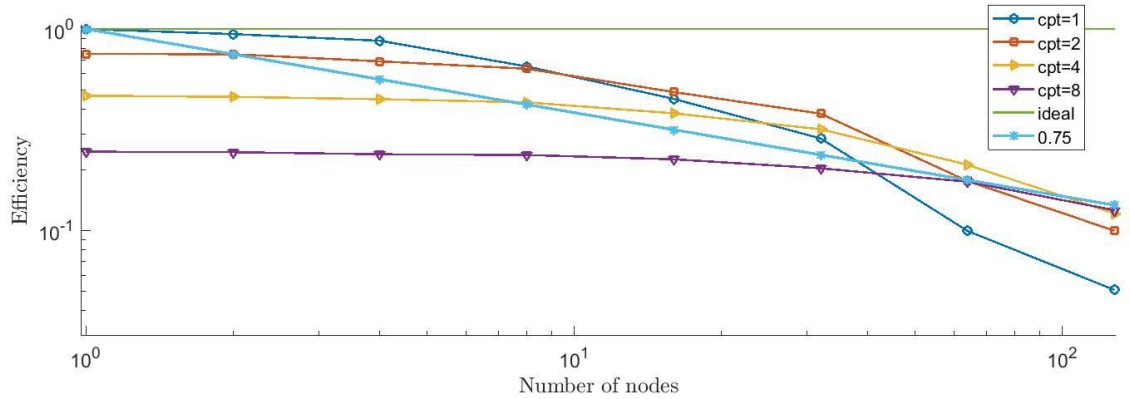
Running on Mahti

- 1404 CPU nodes: 2x AMD Rome 7H12 CPUs (64 cores each)
- 24 GPU nodes: + 4X A100 & NVME
- Required modules: Fortran + Openmpi, openblas
- undersubscribe & spread
- wall time vs # of nodes
- time measurements: mpi_wtime()
- eliminate serial part: $t_{20\ steps}$-$t_{10\ steps}$

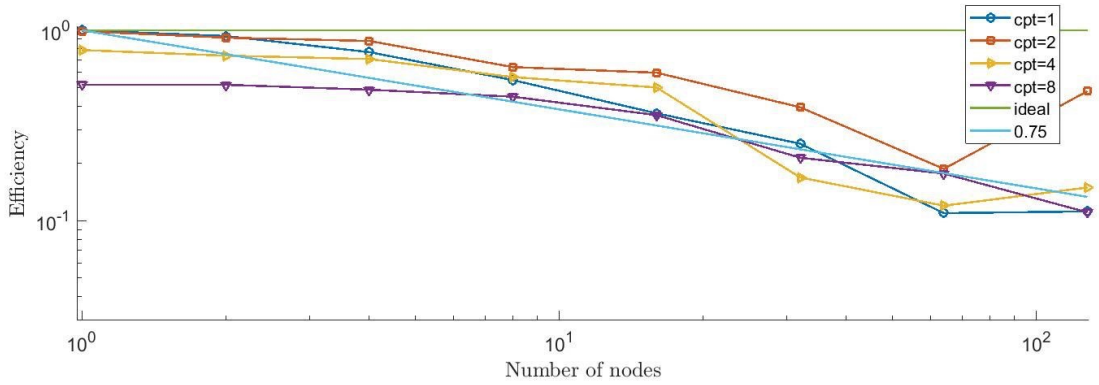
```
*     Read input:  0.474787 seconds |
* Read XYZ files:  3.112545 seconds |
* Neighbor lists:  0.756136 seconds |
*  GAP desc/pred:262.099182 seconds |
    - soap_turbo:135.830049 seconds |
    - lin__turbo: 20.739567 seconds |
    -          2b:  2.305878 seconds |
    -          3b:120.177111 seconds |
    -   core_pot:  0.000000 seconds |
    -         vdw:  0.000000 seconds |
*  MD algorithms: 10.922272 seconds |
*  MPI comms.    :  8.570891 seconds |
    -  pos & vel:  2.144792 seconds |
    - E & F brc.:  6.418340 seconds |
    -  MPI misc.:  0.007758 seconds |
*  Miscellaneous: -0.315245 seconds |

*     Total time:285.620567 seconds |
```

Overall performance and Efficiency ($t_{1\ node}$ / [t x (# of nodes)])



Efficiency vs. # of nodes for different undersubscribing. (on Mahti)

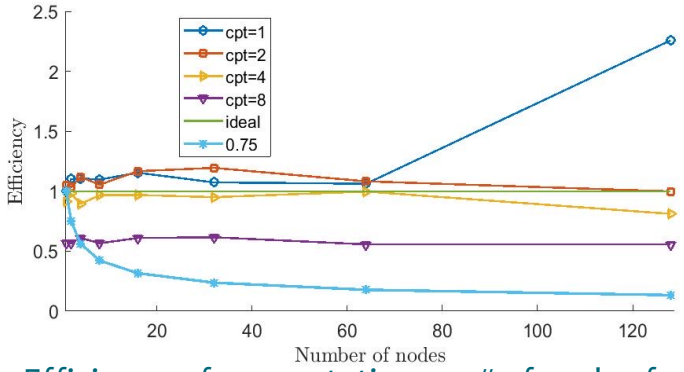


Efficiency vs. # of nodes for different undersubscribing. (on LUMI)

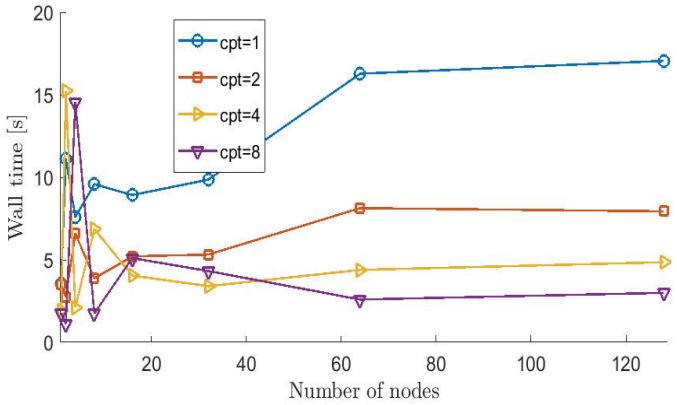
Computation vs. Communication

```
*     Read input:   0.474787 seconds |
* Read XYZ files:   3.112545 seconds |
* Neighbor lists:   0.756136 seconds |
*  GAP desc/pred:262.099182 seconds |
     - soap_turbo:135.830049 seconds |
     - lin__turbo: 20.739567 seconds |
     -          2b:  2.305878 seconds |
     -          3b:120.177111 seconds |
     -    core_pot:  0.000000 seconds |
     -         vdw:  0.000000 seconds |
*  MD algorithms: 10.922272 seconds |
*  MPI comms.   :  8.570891 seconds |
     - pos & vel:  2.144792 seconds |
     - E & F brc.:  6.418340 seconds |
     - MPI misc.:  0.007758 seconds |
*  Miscellaneous: -0.315245 seconds |

*     Total time:285.620567 seconds |
```



Efficiency of computation vs. # of nodes for different undersubscribing.



Time spent in MPI operations vs. # of nodes for different undersubscribing.

Optimizations

- Node & core level optimizations
- Improve Communications:
  - Optimize the MPI
  - Optimize the partition (load balance between MPI tasks & Domain Decomposition)
  - parallel IO, neighbors search, MD step update
  - Reduce the number of MPI tasks.
    - Add OpenMP (for the cpu parts) support
    - Use GPUs to increase the amount operations per process

Porting to GPU

Porting Strategy

- TurboGAP is a FORTRAN code:
  - Keep most of the code
  - As portable as possible
- ~~OpenMP offloading to GPU~~
- Initial port done on Mahti (on nvidia A100 GPUs)
- FORTRAN + CUDA (HIP via hipify)
  - Interoperability done via **iso_c_binding**
    - GPU objects and pointers are of type **c_ptr** in Fortran
  - loop-by-loop approach
    - the rest of the code untouched
    - error checking per variable and loop

Code I

```
k2 = 0
do i = 1, n_sites
do j = 1, n_neigh(i)
k2 = k2 + 1
counter = 0
counter2 = 0
do n = 1, n_max
do np = n, n_max
do l = 0, l_max
if( skip_soap_component(l, np, n) ) cycle
counter = counter+1
do m = 0, l
k = 1 + l*(l+1)/2 + m
counter2 = counter2 + 1

multiplicity = multiplicity_array(counter2)
soap_rad_der(counter, k2) = soap_rad_der(counter, k2) + multiplicity * real( cnk_rad_der(k, n, k2) * &
conjg(cnk(k, np, i)) + cnk(k, n, i) * conjg(cnk_rad_der(k, np, k2)) )
soap_azi_der(counter, k2) = soap_azi_der(counter, k2) + multiplicity * real( cnk_azi_der(k, n, k2) * &
conjg(cnk(k, np, i)) + cnk(k, n, i) * conjg(cnk_azi_der(k, np, k2)) )
soap_pol_der(counter, k2) = soap_pol_der(counter, k2) + multiplicity * real( cnk_pol_der(k, n, k2) * &
conjg(cnk(k, np, i)) + cnk(k, n, i) * conjg(cnk_pol_der(k, np, k2)) )
end do
end do
end do
end do
```

Code II

```
soap_rad_der(:, k2)= soap_rad_der(:, k2)/sqrt_dot_p(i)-soap(:, i)/sqrt_dot_p(i)**3* dot_product( soap(:, i), soap_rad_der(:, k2) )
soap_azi_der(:, k2) = soap_azi_der(:, k2)/sqrt_dot_p(i)-soap(:, i) /sqrt_dot_p(i)**3*dot_product( soap(:, i), soap_azi_der (:, k2) )
soap_pol_der(:, k2) = soap_pol_der(:, k2)/sqrt_dot_p(i)-soap(:, i) /sqrt_dot_p(i)**3*dot_product( soap(:, i), soap_pol_der(:, k2) )
! Transform to Cartesian
if( j == 1 )then
k3 = k2
else
soap_cart_der(1, 1:n_soap :, k2) = dsin(thetas(k2)) * dcos(phis(k2)) * soap_rad_der(1:n_soap, k2) - &
dcos(thetas(k2)) * dcos(phis(k2)) / rjs(k2) * soap_pol_der(1:n_soap, k2) - dsin(phis(k2)) / rjs(k2) * soap_azi_der(1:n_soap, k2)
soap_cart_der(2, 1:n_soap, k2) = dsin(thetas(k2)) * dsin(phis(k2)) * soap_rad_der(1:n_soap, k2) - &
dcos(thetas(k2)) * dsin(phis(k2)) / rjs(k2) * soap_pol_der(1:n_soap, k2) + dcos(phis(k2)) / rjs(k2) * soap_azi_der(1:n_soap, k2)
soap_cart_der(3, 1:n_soap, k2) = dcos(thetas(k2))*soap_rad_der(1:n_soap, k2)+ &
dsin(thetas(k2)) / rjs(k2) *soap_pol_der(1:n_soap, k2)
! MAKE SURE THAT THIS IS CORRECT FOR THE CENTRAL ATOM DERIVATIVES
soap_cart_der(1, 1:n_soap, k3) = soap_cart_der(1, 1:n_soap, k3) - soap_cart_der(1, 1:n_soap, k2)
soap_cart_der(2, 1:n_soap, k3) = soap_cart_der(2, 1:n_soap, k3) - soap_cart_der(2, 1:n_soap, k2)
soap_cart_der(3, 1:n_soap, k3) = soap_cart_der(3, 1:n_soap, k3) - soap_cart_der(3, 1:n_soap, k2)
end if
end do
end do
```

First Target

```
*     Read input:  0.474787 seconds |
* Read XYZ files:  3.112545 seconds |
* Neighbor lists:  0.756136 seconds |
*  GAP desc/pred:262.099182 seconds |
    - soap_turbo:135.830049 seconds |
    - lin__turbo: 20.739567 seconds |
    -          2b:  2.305878 seconds |
    -          3b:120.177111 seconds |
    -   core_pot:  0.000000 seconds |
    -         vdw:  0.000000 seconds |
*  MD algorithms: 10.922272 seconds |
*  MPI comms.   :  8.570891 seconds |
    -  pos & vel:  2.144792 seconds |
    - E & F brc.:  6.418340 seconds |
    -  MPI misc.:  0.007758 seconds |
*  Miscellaneous: -0.315245 seconds |

*     Total time:285.620567 seconds |
```

Descriptors calculations

Energy & Forces prediction
(linear algebra)

SOAP

Descriptors computations

```
call get_soap(n_sites, n_neigh, n_species, species, species_multiplicity, n_atom_pairs, mask, rjs, &
              thetas, phis, alpha_max, l_max, rcut_hard, rcut_soft, nf, global_scaling, &
              atom_sigma_r, atom_sigma_r_scaling, atom_sigma_t, atom_sigma_t_scaling, &
              amplitude_scaling, radial_enhancement, central_weight, basis, scaling_mode, do_timing, &
              do_derivatives, compress_soap, compress_soap_indices, soap, soap_cart_der)

call get_soap_energy_and_forces(soap, soap_cart_der, alphas, delta, zeta, 0.d0, Qs, &
                                n_neigh, neighbors_list, xyz, do_forces, do_timing, &
                                energies, forces, virial)
```

Energy & Forces prediction

C Objects in Fortran

- C pointers, cu/hipBLAS handlers, streams, … are values of **type(c_ptr)** variables in Fortran
- **type(c_ptr):**
  - provided by the **iso_c_binding** module
  - special type of variables which facilitate the Fortran-C interoperability
  - store memory addresses that point to data or objects in a C
  - compiler needs to support 2003 standard
  - straight forward to use:

```
use iso_c_binding
type(c_ptr) :: x
integer(c_int) :: N
```

```
function foo(x, N) bind(C)
    use iso_c_binding
    type(c_ptr) , value :: x
    integer(c_int) :: N
end function
```

```
extern "C" int foo(int *x, int N)
{
….
}
```

GPU Memory Allocation/Deallocation

```
use iso_c_binding
…
type(c_ptr) :: x_d
integer(c_int) :: N
integer(c_size_t) :: size_b
size_b= N*c_double
call gpu_malloc_a(x_d, size_b)
…
call  gpu_free_a(x_d)
```

```
subroutine gpu_malloc_a(x_d, Np) bind(C)
type (c_ptr) :: x_d
Integer (c_size_t),value :: Np
end subroutine

subroutine gpu_free_a(a_d) bind(C)
type(c_ptr) :: a_d
end subroutine
```

```
extern "C" void gpu_malloc_a(void **x_d, size_t Np)
{
    hipMallocAsync( x_d, Np ,0);
    return;
}
extern "C" void gpu_free_a (void **x_d)
{
     hipFreeAsync(*x_d,0);
     return;
}
```

GPU - CPU Data Transfer

```
use iso_c_binding
…
real(c_double) , intent(inout),target:: x_h
call gpu_cpy_htod_a(x_d, c_loc(x_h), size_b)
…
call gpu_cpy_dtoh_a(c_loc(x_h), x_d, size_b)
```

```
subroutine gpu_cpy_htod_a(x_d, x_h, Np) bind(C)
type (c_ptr),value :: x_d,x_h
Integer (c_size_t),value :: Np
end subroutine

subroutine gpu_cpy_dtoh_a(x_h, x_d, Np) bind(C)
type (c_ptr),value :: x_d,x_h
integer (c_size_t),value :: Np
end subroutine
```

```
extern "C" void gpu_cpy_htod_a(void *x_d, void *x_d ,size_t Np)
{
    hipMemcpyAsync( x_d, x_h, Np, hipMemcpyHostToDevice);
    return;
}
extern "C" void gpu_cpy_htod_a(void *x_d, void *x_d ,size_t Np)
{
    hipMemcpyAsync( x_h, x_d, Np, hipMemcpyDeviceToHost);
    return;
}
```

BLAS Calls

```
type(c_ptr) :: bhandle
call gpu_create_handle(bhandle)
…
call gpu_blas_mvmul_n(bhandle, x_d,  v_d, c_d, nX, nY)
```

```
subroutine gpu_create_handle(bhandle) bind(C)
    type (c_ptr) :: bhandle
end subroutine
subroutine gpu_blas_mvmul_n(bhandle, x_d,  &
                            v_d, c_d, nX, nY) bind(C)
    type (c_ptr),value :: bhandle, x_d v_d, c_d
    integer (c_int),value :: nX, nY
end subroutine
```

```
extern "C" void gpu_create_handle(hipblasHandle_t *bhandle)
{
    hipblasCreate( bhandle);
    return;
}
extern "C" void gpu_cpy_htod_a(hipblasHandle_t bhandle, double *x_d, double *v_d, double *c_d, int nX, int nY)
{
    const double a=1, b=0;
    const double *alpha= &a, *beta= &b;
    hipblasDgemv( bhandle, HIPBLAS_OP_N, nX, nY, alpha,x_d, nX,  v_d, 1, beta, c_d, 1);
    return;
}
```

Kernels Calling

```
…
call gpu_pow(x_d, y_d, zeta, N)
…
```

```
subroutine gpu_pow(x_d, y_d, zeta, N) bind(C)
    type (c_ptr),value :: x_d, y_d
    integer (c_int),value :: N
    real (c_double),value :: zeta
end subroutine
```

```
__global__ void array_pow(double *x_d, double *y_d, int N)
{
    int idx=threadIdx.x + blockIdx.x*blockDim.x;
    if (idx<N) {
        double loc_x=x_d[idx];
        y_d[idx]=pow(loc_x, zeta);
    }
}
extern "C" void gpu_pow(double *x_d, double *y_d, double zeta, int N)
{
    array_pow<<<(N+tpb-1)/tpb,tpb>>>(x_d,y_d,zeta,N);
    return;
}
```

Single GPU Performance I. CPU vs GPU

Pure CPU	CPU+GPU
Mahti	
- soap_turbo: 141.785 seconds - multi_soap: 105.499 seconds	- soap_turbo: 41.232 seconds - multi_soap: 5.079 seconds
LUMI	
- soap_turbo: 70.735 seconds - multi_soap: 52.192 seconds	- soap_turbo: 22.039 seconds - multi_ soap: 3.990 seconds

SingleGPU Performance II. Top 10 Kernels

A100			MI250X (1GCD)		
Kernels 1.52173856225 s			Kernels 1.863860614 s		
Name	TotalDurationNs	%	Name	TotalDuration Ns	%
cuda_get_soap_der_one	378912902	24.9	cuda_get_soap_der_one	350280919	18.7
cuda_soap_forces_virial_two	153886417	10.1	cuda_get_derivatives_new_new	310049084	16.6
cuda_get_derivatives_new_new	150755929	9.9	cuda_get_cnk_one_new_new	156669918	8.4
gpu_pow	115433949	7.6	gemvn_kernel	133860688	7.1
cuda_get_soap_p	111639111	7.3	gpu_pow	116008318	6.2
naive_transpose_soap_rad_azi_pol	96456682	6.3	cuda_soap_forces_virial_two	11576254	6.2
cuda_get_cnk_one_new_new	77005589	5.0	cuda_get_soap_p	106622502	5.7
cuda_get_soap_der_thr_one	66496200	4.4	naive_transpose_soap_rad_azi_pol	97049510	5.2
cuda_get_exp_coeff_der_one	66408620	4.3	Cijk_Alik_Bljk	75350526	4.0
cuda_get_soap_der_two_two	54276453	3.	Cijk_Ailk_Bjlk	263549	3.4

Discussion

Summary & Take Home message

- FORTRAN + CUDA/HIP
  - **iso_c_binding:** GPU objects & pointers as **c_ptr**
  - reasonable portability (hipify / **H**eader **O**nly **P**orting)
  - loop-by-loop approach
    - the rest of the code untouched
    - error checking per variable and loop
  - initial performance 14-16 x speed-up (on single GPU)
  - high overhead from cpu – gpu transfers & no overlapping
- port as much as possible
- add OpenMP support (for the CPU components), improve MPI and partitioning
- improve the overall code design
- test the all potentials

Aknowledgements

- Academy of Finland for granting the funds
- ExaFF consortium members who prepared the applications
- Technical slides from Miguel Caro
- LUST, HPE & AMD

Cristian Achim

CSC – IT Center for Science Ltd.
Application Scientist
cristian-vasile.achim@csc.fi


facebook.com/CSCfi


twitter.com/CSCfi


linkedin.com/company/csc---it-center-for-science


github.com/CSCfi