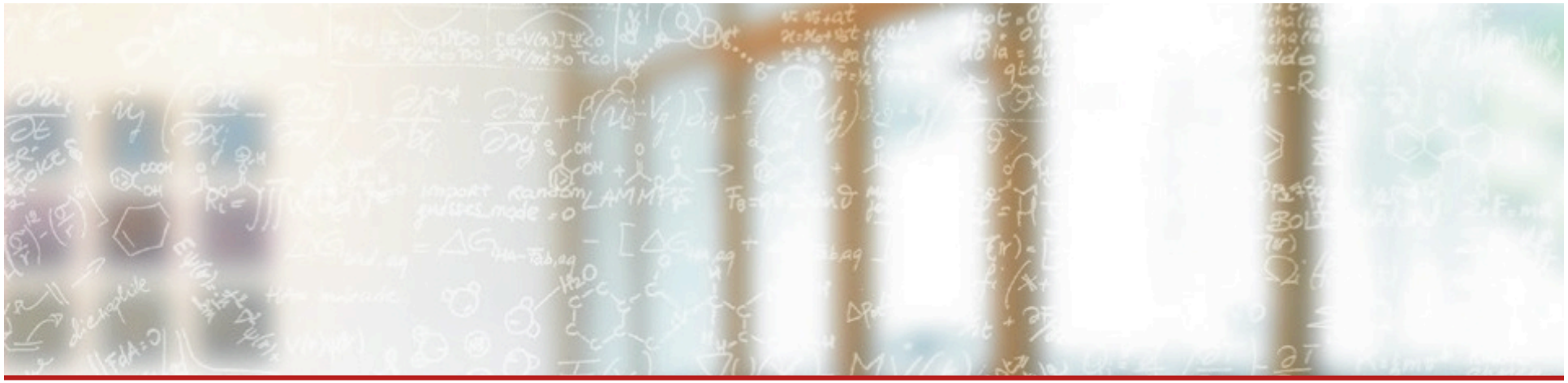
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Lugano GPU Hackathon 2017

Ste||ar Group

Octotiger

CUDA integration of FMM

- Convert FMM kernels to cuda
  - Multipole-Multipole
    - Working, and giving **almost** correct results
    - (2 dec places)
  - KNL refactoring for SIMD types Maps well to CUDA
  - Reused most of kernel with minor tweaks (SIMD types)
    - Vc SIMD library for AVX – reduced back to scalar
    - Needed many annoying cruft #define for CPU/GPU operation
  - NVCC C++14 causes most trouble (device-STL)

More FMM progress

Monopole-Monopole

- Modified to use SIMD style approach
  - Vc library (KNL implemented)
- Not CUDA implemented – but CUDA ready
  - As per Multipole kernel, needs a minor rewrite for CUDA operation
  - First kernel experience enough to know what to do
- + other 2 TDB

CUDA compilers

- Octotiger working with CUDA using
  - NVCC
    - Some C++ features missing
  - Clang
    - Totally awesome with full C++14 support (C++17)
- Many own goals with clang cuda, NVCC, etc
  - Wasted a lot of time getting fancy features to work when we can live without them for V1.
  - For first Octotiger integration we should have kept it simple – but future work will be easier
- Daint + laptop – same toolchain :)

Work in progress

- Each FMM call has overhead of N milliseconds
  - Too slow to be useful since CPU does same job in N-1 milliseconds
- Must combine FMM calls from M patches
  - Modify kernel to take array of FMM data
  - Increase num threads from 8^3 to M.8^3
  - Collect data from each patch and place into queued FMM batch executor
    - Launch one kernel for M FMM ops

```
template <typename R, typename C, int N, typename...Args>
class interaction_batcher<R(C::*)(Args...), N>
```

Next Steps

- Octotiger is asynchronous at all levels
  - Batching of FMM requires some redesign
  - M FMM kernels must be gathered
  - Tasks working on those must yield
- GPU execution can return future
  - hpx::future<void> connected to CUDA kernel via stream
    - Working and tested
    - Nice integration with HPX runtime
- When hpx::future<batch GPU kernel> ready
  - my_gpu_future.then(do more stuff);

Conclusion

- Hackathon a success
  - Unfortunately wasted too much time at start
- Inside Octotiger
  - We have cuda FMM kernel essentially working
  - We have cuda futures working
  - We know how to proceed with delivering more work to GPU
- After Hackathon
  - Continue work on kernels and batching/tasking
  - Look for more places to offload GPU work
  - Experiment with fancy Cuda Clang C++ integration