# PERFORMANCE PORTABILITY STRATEGIES FOR GRID C++ EXPRESSION TEMPLATE

Meifeng Lin

Brookhaven National Laboratory

Lattice 2017, Granada, Spain, June 18-24, 2017

## COLLABORATORS

|  |  |
|---|---|
| Peter Boyle | University of Edinburgh |
| Kate Clark | NVIDIA |
| Carleton DeTar | University of Utah |
| Verinder Rana | Brookhaven National Laboratory |
| Alejandro Vaquero | University of Utah |

## MOTIVATION

### Exascale

- US DOE is working on delivering exascale systems in the next couple of years (2021-2023).
- Architectures are expected to be diverse, likely to have heterogeneity, complex memory hierarchy, multiple levels of parallelism, etc..
- USQCD and its partners are working on next-generation software for lattice QCD as part of the US Exascale Computing Project (ECP).

  See Carleton DeTar, "Lattice QCD Application Development within the US DOE Exascale Computing Project", 15:00 Thursday

### ECP Software Requirements

- **Efficiency**: Should be able to efficiently exploit the expected multiple levels of parallelism on the exascale architectures. Need to conquer the communication bottleneck.
- **Flexibility**: Should be flexible for the users to implement different algorithms and physics calculations, and can provide easy access to multi-layered abstractions for the users.
- **Performance Portability**: Should be portable to minimize code changes for different architectures while maintaining competitive performance.

## GRID

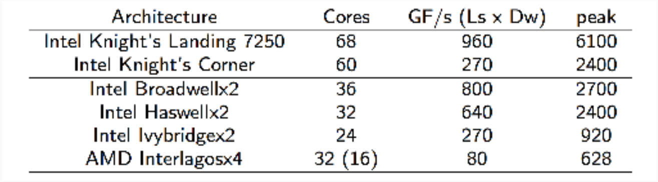
- A data parallel C++ mathematical object library. https://github.com/paboyle/Grid
- Developed by Peter Boyle, Guido Cossu, Antonin Portelli and Azusa Yamaguchi at the University of Edinburgh.
- Written in C++11. Extensive use of templates to allow for high-level abstractions.

```
GridCartesian     Grid(latt_size,simd_layout,mpi_layout);

LatticeColourMatrix A(&Grid);
LatticeColourMatrix B(&Grid);
LatticeColourMatrix C(&Grid);

C = A * B
```

- Expression template makes this possible.
- Data layout designed for SIMD architectures with different SIMD widths. Intrinsics/compiler vectorization/OpenMP directives may be used for vectorization depending on target.
- Many architectures supported with good performance.



| Architecture | Cores | GF/s (Ls x Dw) | peak |
|---|---|---|---|
| Intel Knight's Landing 7250 | 68 | 960 | 6100 |
| Intel Knight's Corner | 60 | 270 | 2400 |
| Intel Broadwellx2 | 36 | 800 | 2700 |
| Intel Haswellx2 | 32 | 640 | 2400 |
| Intel Ivybridgex2 | 24 | 270 | 920 |
| AMD Interlagosx4 | 32 (16) | 80 | 628 |

P. Boyle

## GRID C++ EXPRESSION TEMPLATE

### Some Background

- GPU is not among the supported architectures at the moment.
- Initial GPU porting effort started last year using OpenACC.
  - Ran into many issues due to Grid's complex data structures. $\hookrightarrow$ deep copy
  - PGI compiler did not sufficiently support C++11 code.
  - STL not supported on GPUs.
  - Porting whole Grid turned out to be rather difficult.
- Proof-of-concept studies using stripped-down version of Grid expression template (ET) done last summer by P. Boyle and ML.

### Grid ET

- $\sim 200$ lines of self-contained code, provided by P. Boyle.
- Arithmetic operations contained in the recursive `eval` function
  $\hookrightarrow$ `for` loop is target to be offloaded to the GPU.

```
template <typename Op, typename T1,typename T2> inline Lattice<obj> & operator=(const
   LatticeBinaryExpression<Op,T1,T2> expr)
{
  int _osites=this->Osites();
  for(int ss=0;ss<_osites;ss++){
    _odata[ss] = eval(ss,expr);
  }
  return *this;
}
```

## DIFFERENT APPROACHES STUDIED

OpenACC/OpenMP

- Pros: Directives-based approach; Easy to add to existing code; Portable across different platforms.
- Cons: Lack of deep-copy support; Use in C++ code non-trivial; Dependent on compiler; Developer has little control.

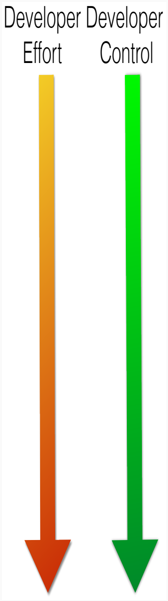
Just-In-Time: Jitify

- New JIT header library being developed at NVIDIA.

  See GTC2017 talk - Ben Barsdell, Kate Clark "Jitify: CUDA C++ Runtime Compilation Made Easy"
- Pros: No need for CUDA extensions (though available). CPU and GPU execution policies can be present simultaneously.
- Cons: Runtime compilation. Kernel functions need to be given in header files.

CUDA

- Pros: Mature programming model for NVIDIA GPUs. C++ support is steadily improving. Easy to control for performance.
- Cons: Need to write some CUDA kernels; Some code branching unavoidable. Supports NVIDIA GPUs only. Need to declare all host device functions.



## COMPARISON OF DIFFERENT IMPLEMENTATIONS

### Kernel

OpenACC

```
#pragma acc parallel loop independent copyin(expr[0:1])
 for(int ss=0;ss<_osites;ss++){
    _odata[ss] = eval(ss,expr);
 }
```

OpenMP

```
#pragma omp target device(0) map(to: expr) map(tofrom:_odata[0:_osites])
  {
  #pragma omp teams distribute parallel for
     {
     for (int i=0; i<_osites; i++)
        _odata[ss] = eval(ss,expr);
     }
   }
```

Jitify

```
parallel_for(policy, 0, _osites,
              JITIFY_LAMBDA( (_odata,expr),
              _odata[i]=eval(i,expr); ));
```

CUDA

```
 template<class Expr, class obj> __global__
  void ETapply(int N,obj *_odata,Expr Op)
  {
    int ss = blockIdx.x;
    _odata[ss]=eval(ss,Op);
  }
 LatticeBinaryExpression<Op,T1,T2> temp = expr;
 ETapply< decltype(temp), obj > <<<_osites,1>>>((int)_osites,this->_odata,temp);
```

## OTHER CODE CHANGES/COMPARISONS

- OpenACC
  - Need to specify device routines with `#pragma acc routine`. Defined in OFFLOAD.
  - Need PGI's Unified Virtual Memory (UVM) support for data management.
  - Choose target at compile time

```
[GPU] pgc++ -acc -ta=tesla:managed --c++11  -O3 main.cc -o gpu.x
[CPU] pgc++ -acc -ta=multicore --c++11-O3 main.cc -o cpu.x
```

- OpenMP
  - Similar to OpenACC, but no compiler UVM support yet. So code is not working yet.
- Jitify
  - Use managed memory allocator for UVM support. Execution policy defined in main program.

```
 static const Location ExecutionSpaces[] = DEVICE;
 policy = ExecutionPolicy(location);
```

- CUDA
  - Customized allocator: aligned allocator for CPUs, managed allocator for GPUs.

```
#ifdef GPU
    cudaMallocManaged((void **)&ptr, __n*sizeof(_Tp));
#elif defined(AVX512)
    ptr = (pointer) _mm_malloc(__n*sizeof(_Tp), 64); //changes with the target architecture
#elif ...
```

  - OFFLOAD macro needed for functions on both host and device

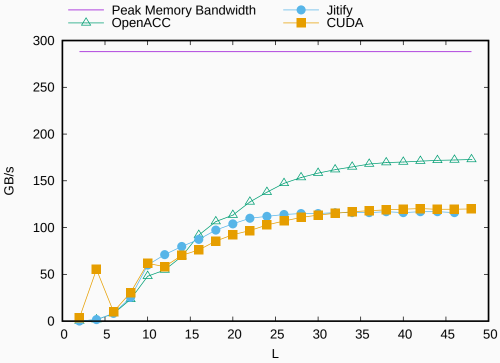
```
#ifdef __NVCC__
#define OFFLOAD __host__ __device__
#elif defined (_OPENACC)
#define OFFLOAD _Pragma(”acc routine seq”)
#else
#define OFFLOAD
#endif
```

## SU(3)xSU(3) STREAMING TEST

- Wrote own SU(3) class with AoS data layout (A. Vaquero)
- Fed into Grid ET

```
Lattice<Su3f> z(&grid);
Lattice<Su3f> x(&grid);
Lattice<Su3f> y(&grid);
for(int i=0;i<Nloop;i++) {
      z=x*y;
}
```

- Performance comparison with default setting (no tuning of thread/block numbers). NVIDIA GTX 1080 (Pascal Gaming Card)

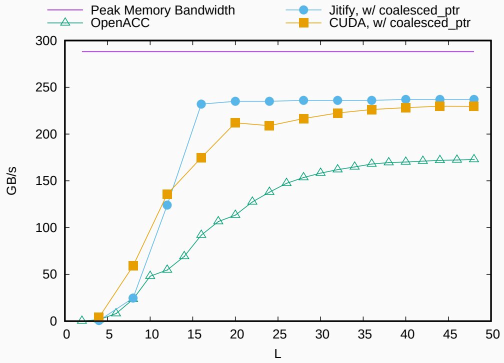


## SU(3)xSU(3) streaming test

- Wrote own SU(3) class with AoS data layout (A. Vaquero)
- Fed into Grid ET

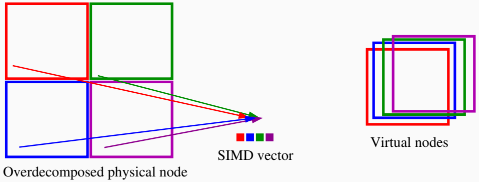
```
Lattice<Su3f> z(&grid);
Lattice<Su3f> x(&grid);
Lattice<Su3f> y(&grid);
for(int i=0;i<Nloop;i++) {
      z=x*y;
}
```

- Performance comparison with default setting (no tuning of thread/block numbers). NVIDIA GTX 1080 (Pascal Gaming Card)

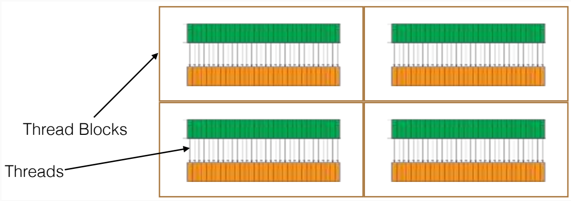


## MAPPING SIMD DATA LAYOUT ONTO GPUS

- Poor performance due to lack of memory coalescing with the AoS data layout
- Can be overcome by using a `coalesced_ptr` class (K. Clark)
  - Transforms AoS into AoSoAoS
  - Performance boost by a factor of 2
- Grid's native SIMD vector layout can be used to ensure coalescence without coalesced_ptr.



- Each GPU thread within a thread block processes one element of the vector. Thread blocks map to outer sites.



## CODE SNIPPETS

- Since the top-level data structures are of vector types, some “hacking” is needed to make different threads process different elements of the vector.
- Brookhaven Hackathon: ML, Alejandro Vaquero, Mathias Wagner (mentor).
- Make each thread `eval` one element of the vector, extracted through `extractS`.

```
//C++14 and CUDA 9 needed to make this work
 template<class obj> OFFLOAD inline auto evalS(const unsigned int ss, const Lattice<obj> arg,
      const int tIdx) //-> decltype(typename obj::scalar_object)
 {
   typedef typename obj::scalar_object sObj;
   auto sD = extractS<obj,sObj>(arg._odata[ss], tIdx);
   return (sObj) sD;
 }
```

- Put the results back to form the vector again after `eval`.

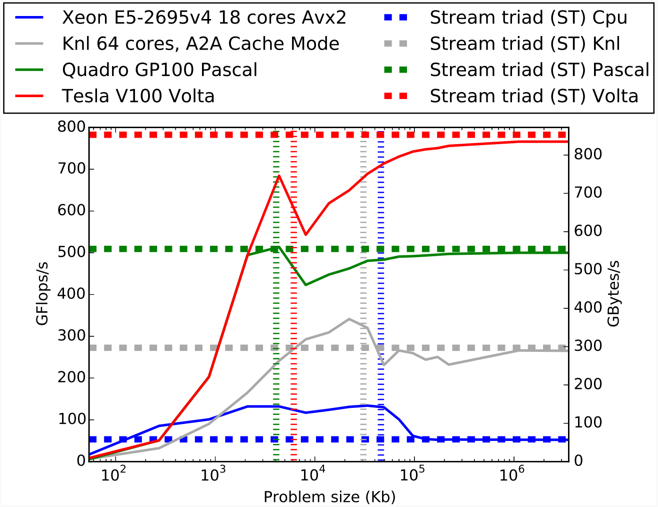
```
template<class Expr, class obj> __global__
void ETapply(int N,obj *_odata,Expr Op)
{
  if (blockIdx.x < N) {
    typedef typename obj::scalar_object sObj;
    auto sD = evalS(blockIdx.x,Op,threadIdx.x);
    mergeS(_odata[blockIdx.x], sD, threadIdx.x);
  }
}
```

- Outer sites become thread blocks; inner sites become threads.

```
ETapply<decltype(temp), obj> < < <_osites,_isites> > > ((int)_osites,this->_odata,temp);
```

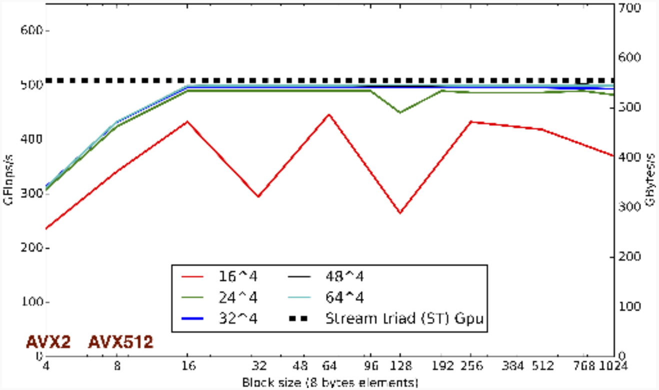
## SU(3)xSU(3) PERFORMANCE

Same code. Performance saturates STREAM Triad results on multiple architectures.



## BLOCK SIZE DEPENDENCE

- How big do we need to make the blocks?
- Twice the AVX512 width (1024 bits) already saturates the performance.



Tests on NVIDIA Quadro GP100

## CONCLUSIONS

- We have successfully ported an SU(3)xSU(3) miniapp based on Grid's expression template to the GPU using OpenACC, CUDA and Jitify.
- Unified Virtual Memory is used in all the implementations to simplify data management.
- The performance of the GPU code depends heavily on the memory coalescence of the data layout.
- Performance can be improved with a `coalesced_ptr` class for the AoS layout.
    - Did not work for the OpenACC implementation due to issue with `std::complex`.
- Grid's native SIMD vector layout maps well onto GPUs, and performance is saturated across multiple architectures without resorting to `coalesced_ptr`.

### Next Steps

- We will start working on other parts of Grid that live outside of the expression template.
- Porting a Dslash kernel to GPUs is an obvious next step.

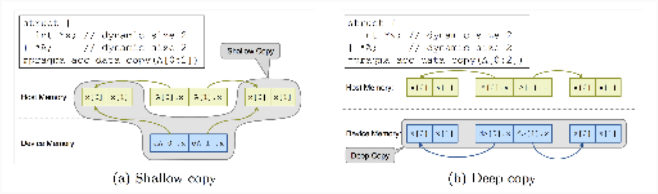
## ACKNOWLEDGMENTS

- We thank Mathias Wagner (NVIDIA) for his help during the recent hackathon.
- This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, in support of the nation’s exascale computing imperative.
- Part of this research was carried out at the Brookhaven Hackathon 2017. Brookhaven Hackathon is a collaboration between and used resources of Brookhaven National Laboratory, University of Delaware, Stony Brook University, and the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory.

## Backup Slides

## DEEP COPY

- Current OpenACC standard does not support *deep copy*.
- For user-defined data types (arrays of structures), simple copy will result in incorrect pointer dereferencing.



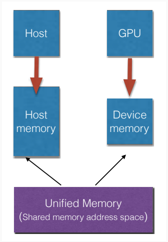
```
struct {
  int *x; // dynamic size 2
} *A;     // dynamic size 2
#pragma acc data copy(A[0:2])
```

```
struct {
  int *x; // dynamic size 2
} *A;     // dynamic size 2
#pragma acc data copy(A[0:2])
```

(a) Shallow copy (b) Deep copy

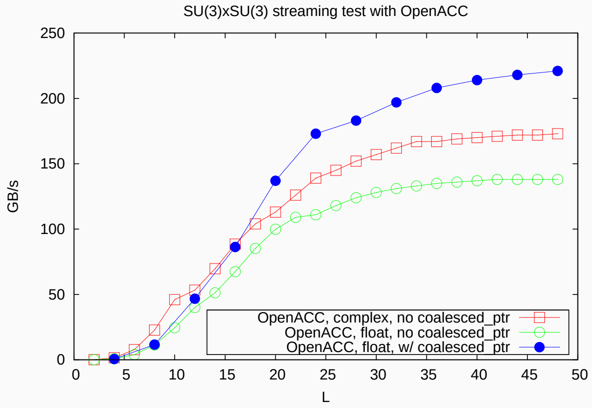
Source: www.openacc.org

- Work around: PGI compiler provides the ”managed” option to use NVIDIA unified memory.
- Pre-Pascal: The maximum size of the unified memory space is limited by the GPU memory.
- Pascal: Allows oversubscription of GPU memory. Unified memory can be as big as host memory.



## OPENACC WITH COALESCED_PTR

- Code would not compile if coalesced_ptr is used. Possible compiler bug.
- Comparison using real numbers also shows significant improvement with the coalesced pointer.


SU(3)xSU(3) streaming test with OpenACC
GB/s
L
OpenACC, complex, no coalesced_ptr
OpenACC, float, no coalesced_ptr
OpenACC, float, w/ coalesced_ptr