HMC, high level structures and architecture support in Grid

Guido Cossu

in collaboration with

P. Boyle, A. Portelli, A. Yamaguchi

GRID library

Started from Peter’s effort to design a modern C++ library for Cartesian mesh problems

Design goals
- performance portability
- zero code replication

Grid github pages github.com/paboyle/Grid
(partial) documentation paboyle.github.io/Grid/
CI: Travis, TeamCity https://ci.cliath.ph.ed.ac.uk/

Source code: C++11, autotools

+ HADRONS physics measurement framework based on Grid (A. Portelli)
Intense work in progress, production stage next year

Harness the power of generic programming

- Define algorithms for generic types
- Templates & template metaprogramming
- Define general interfaces and let the compiler do the hard job
- Basic types will mask the architecture from high level classes
- Enters C++11
  - type inference
  - new standard library, metaprogramming improvements
  - type traits
  - variadic templates
  - ...
- Write code once!

vSIMD, basic portable vector types

Define performant classes `vRealF, vRealD, vComplexF, vComplexD`.

Here very simplified, actual implementation use extensively C++11 type traits.

```
#if defined (AVX1) || defined (AVX2)
         typedef __m256 SIMD_Ftype;
#endif
#if defined (SSE2)
         typedef __m128 SIMD_Ftype;
#endif
#if defined (AVX512)
         typedef __m512 SIMD_Ftype;
#endif
template <class Scalar_type, class Vector_type>
class Grid_simd {
         Vector_type v;
         // Define arithmetic operators
         friend inline vRealD operator + (vRealD a, vRealD b);
         friend inline vRealD operator - (vRealD a, vRealD b);
         friend inline vRealD operator * (vRealD a, vRealD b);
         friend inline vRealD operator / (vRealD a, vRealD b);
         static int Nsimd(void) { return sizeof(Vector_type)/sizeof(Scalar_type);
}
typedef Grid_simd<float, SIMD_Ftype> vRealF;
```

GRID library: architecture support

Current support

- SSE4.2
- AVX, AVX2 (e.g. Intel Haswell, Broadwell, AMD Ryzen)
- AVX512F (Intel KNL, Skylake)
- QPX (BlueGene/Q), experimental
- NEON ARMv8 (on the way, thanks to Nils Meyer from Regensburg University)
- Generic vector width support

Plans for

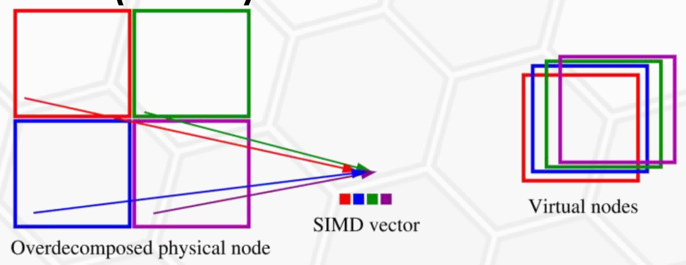
- CUDA threads (Nvidia GPUs) (Alexander Wennersteen & ECP collaboration)
- ARM SVE (Scalable Vector Extensions, Fujitsu post-K)

Each file is O(500) lines of code describing the lowest level operations on the vector types

- Real/Complex algebra
- Load/Store, vstream
- Rotations
- Permutations
- Precision change

GRID parallel library

- Geometrically decompose cartesian arrays across nodes (MPI)
- Subdivide node volume into smaller virtual nodes
- Spread virtual nodes across SIMD lanes
- Use OpenMP+MPI+SIMD to process *conformable array* operations
- Same instructions executed on many nodes, each node operates on `Nsimd` virtual nodes
- Conclusion: modify data layout to align data parallel operations to SIMD hardware
- Conformable array operations are simple and vectorise perfectly



Grid data parallel template library

- Opaque C++11 containers hide data layout from user
- Automatically transform layout of arrays of mathematical objects using vSIMD scalar, vector, matrix, higher rank tensors.

General linear algebra with vector types

```
vRealF, vRealD, vComplexF, vComplexD

template<class vtype> class iScalar
{
      vtype _internal;
};
template<class vtype,int N> class iVector
{
      vtype _internal[N];
};
template<class vtype,int N> class iMatrix
{
      vtype _internal[N][N];
};
```

- Defines matrix, vector, scalar site operations
- Internal type can be SIMD vectors *or* scalars

```
LatticeColourMatrix A(Grid);
LatticeColourMatrix B(Grid);
LatticeColourMatrix C(Grid);
LatticeColourMatrix dC_dy(Grid);
C = A*B;
const int Ydim = 1;
dC_dy = 0.5*Cshift(C,Ydim, 1 ) - 0.5*Cshift(C,Ydim,-1 );
```

- *High-level* data parallel code gets 65% of peak on AVX2
- Single data parallelism model targets BOTH SIMD and threads efficiently.

QCD types example:

```
template<typename vtype> using
iLorentzColourMatrix =
iVector<iScalar<iMatrix<vtype, Nc> >, Nd > ;
```

GRID library: HMC summary

- Actions
  - Gauge: Wilson, Symanzik, Iwasaki, RBC, DBW2, generic Plaquete + Rectangle
  - Fermion: Two Flavours, One Flavour (RHMC), Two Flavours Ratio, One Flavour Ratio. All with the EO variant.
  - Kernels: Wilson, Wilson TM, generalised DWF (Shamir, Scaled Shamir, Mobius, Zmobius, Overlap, …)
  - Scalar Fields
- Integrators: Leapfrog, 2nd order minimum-norm (Omelyan), force gradient, + implicit versions
- Fermion representations
  - Fundamental, Adjoint, Two-index symmetric, Two-index antisymmetric, and all possible mixing of these. Any number of colours. All fermionic actions are compatible.
- Stout smeared evolution with APE kernel (for SU(3) fields). Any action can be smeared.
- Serialisation: XML, JSON
- Algorithms: GHMC, RMHMC, LAHMC, density of states LLR easily implemented
- File Formats: Binary, NERSC, ILDG, SCIDAC (for confs). MPI-IO for efficient parallel IO

Some features inherited by IroIro++

GRID library: HMC design

Need to support a large variety of actions, e.g.

- Scalar field theories
- Fermions in different representations (in any combination)
- Different algorithms (HMC, RMHMC, LAHMC, DoS)

Abstract away invariant concepts by templating

```
template <class Implementation,
          template <typename, typename, typename> class Integrator,
          class RepresentationsPolicy = NoHirep, class ReaderClass = XmlReader>
class HMCWrapperTemplate: public HMCRunnerBase<ReaderClass>


template <class FieldImplementation, class SmearingPolicy, class RepresentationPolicy>
class Integrator
```

GRID library: HMC design

```
template <class Implementation,
          template <typename, typename, typename> class Integrator,
          class RepresentationsPolicy = NoHirep, class ReaderClass = XmlReader>
class HMCWrapperTemplate: public HMCRunnerBase<ReaderClass>
```

Implementation

Boundary conditions
Field types
Momenta generation
Field update
Force projection

Representation

Update representation
Force Algebra projection
Representation to Fundamental projection

C++11 `std::tuples`
The action set is a collection of actions
Need to manage different types
Dispatches the correct function when requested

GRID library: HMC highest level design

User point of view, three usage models:

- Pure C++ code
- Pure C++ and serialisation (all parameter classes are serializable), XML or JSON
- Full control via XML using factories (highest level ~20 lines of code)

Module structure + resources class

GRID library: pure C++

```cpp
int main(int argc, char **argv) {
using namespace Grid;
using namespace Grid::QCD;

Grid_init(&argc, &argv); GridLogLayout();

// Typedefs to simplify notation
typedef GenericHMCRunner<MinimumNorm2> HMCWrapper;
HMCWrapper TheHMC;

// Grid from the command line
TheHMC.Resources.AddFourDimGrid("gauge");
// Possibile to create the module by hand
// hardcoding parameters or using a Reader

// Checkpointer definition
CheckpointerParameters CPparams;
CPparams.config_prefix = "ckpoint_lat";
CPparams.rng_prefix = "ckpoint_rng";
CPparams.saveInterval = 1;
CPparams.format = "IEEE64BIG";
TheHMC.Resources.LoadNerscCheckpointer(CPparams);

RNGModuleParameters RNGpar;
RNGpar.serial_seeds = "1 2 3 4 5";
RNGpar.parallel_seeds = "6 7 8 9 10";
TheHMC.Resources.SetRNGSeeds(RNGpar);

// Construct observables
typedef PlaquetteMod<HMCWrapper::ImplPolicy> PlaqObs;
typedef TopologicalChargeMod<HMCWrapper::ImplPolicy> QObs;
TheHMC.Resources.AddObservable<PlaqObs>();
TheHMC.Resources.AddObservable<QObs>();
/////////////////////////////////////////////////////////////
// Collect actions
RealD beta = 5.6 ;
WilsonGaugeActionR Waction(beta);
ActionLevel<HMCWrapper::Field> Level1(1);
Level1.push_back(&Waction);
TheHMC.TheAction.push_back(Level1);
/////////////////////////////////////////////////////////////

// HMC parameters are serialisable
TheHMC.Parameters.MD.MDsteps = 20;
TheHMC.Parameters.MD.trajL = 1.0;

TheHMC.ReadCommandLine(argc, argv);
TheHMC.Run();
Grid_finalize();
} // main
```

GRID library: pure C++ and serializers

```
namespace Grid{
struct FermionParameters: Serializable {
GRID_SERIALIZABLE_CLASS_MEMBERS(FermionParameters,
         int, Ls,
         double, mass,
         double, M5,
         double, b,
         double, c,
         double, StoppingCondition,
         int, MaxCGIterations,
         bool, ApplySmearing);
};


struct MobiusHMCParameters: Serializable {
GRID_SERIALIZABLE_CLASS_MEMBERS(MobiusHMCParameters,
         double, gauge_beta,
         FermionParameters, Mobius)
```

```
typedef Grid::JSONReader JSONSerialiser;
typedef Grid::XMLReader XMLSerialiser;
HMCWrapper TheHMC;
JSONSerialiser Reader(TheHMC.ParameterFile);

MobiusHMCParameters MyParams(Reader);
```

Summary

Architecture support

- Wide variety of architectures
- Fairly easy implementation: new vector architecture support essentially needs just reading the ISA
- GPU support, work in progress using CUDA by Alexander Wennersteen + ECP collaboration

Grid HMC fairly flexible

- Accommodates a variety of theories/algorithms
- Mixed representations implemented by aggregating types, `std::tuples`

Currently number of dimensions and colours are hardcoded
Plan: create "LatticeTheory" classes that define these and template the implementations to mix gauge theories.

HMC, high level structures and architecture support in Grid

Guido Cossu

in collaboration with

P. Boyle, A. Portelli, A. Yamaguchi