THE UNIVERSITY *of* EDINBURGH
School of Physics
and Astronomy

Grid

a next generation data parallel C++ library

Peter Boyle, **Guido Cossu**,
Antonin Portelli, Azusa Yamaguchi

Work funded as an Intel Parallel Computing Centre

September 8th 2016, Edinburgh
DiracDay 2016

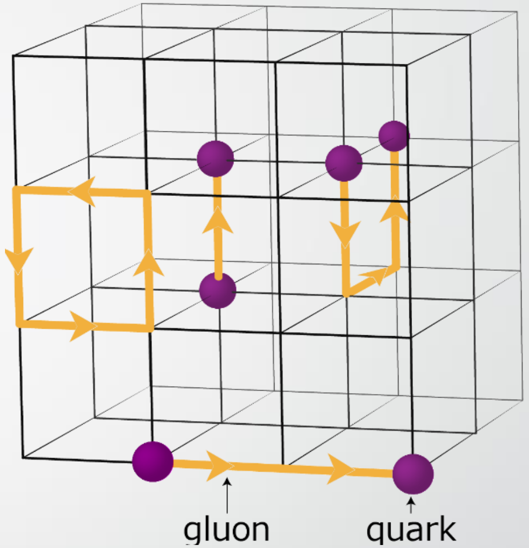
Lattice Gauge Theories (LGT): a primer

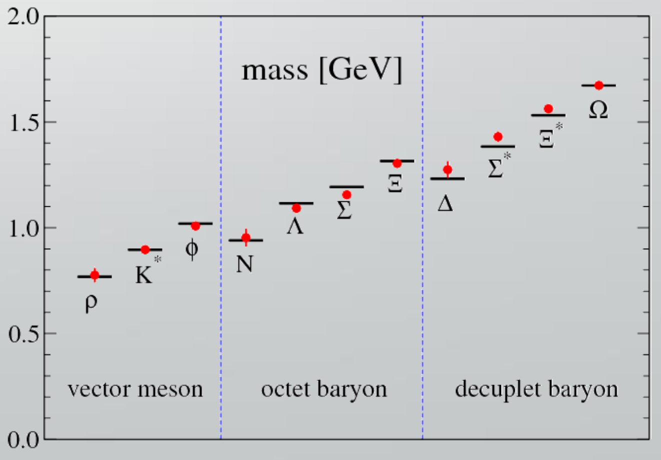
Lattice QCD is the only known non-perturbative regularisation of QCD

Path integral formulation of the expectation values reduced to the computation to few steps:

- Generate configurations with a Boltzmann weight (equivalent to a thermal ensemble at equilibrium)
- Measure and average the observables on these ensembles
- Take the appropriate limits (continuum, infinite volume, realistic quark masses, …)

Including fermions requires the inversion of the Dirac operator, a very large sparse matrix





LGT & HPC

- Current requirements are O(Petaflop) scale machines for O(1-2 years) for accurate measurements including charm quarks (i.e. fine lattices)
- Many observables, many different discretisations, theories, ...
  - Codebases: O(10^{5-6}) lines
- Current HPC: architecture proliferation & large parallelism hierarchy
  - SIMD (SIMT), threading, multi-node
- Need for a high level code that is mostly unaware of the underlying architecture (portability) while preserving performance

Tackle vectorization: SIMD

A technologically cheap way to accelerate code

Isn't there an easier way to get good performance on KNL and Haswell/Skylake?

Text book computing science: (e.g. Hennessy & Patterson)

- Code optimisations should expose *spatial data reference locality*
- Code optimisations should expose *temporal data reference locality*

SIMD brings a new level of restrictiveness that is much harder to hit

- Code optimisations should expose *spatial operation locality*

Aren't we going to have to make it easier to use 128/256/512/???? bit SIMD?

Plan:

- Clean slate reengineer a Lattice QCD interface to exploit all forms of parallelism effectively, MPI & OpenMP & SIMD (SIMT)
- Keep an open strategy for OpenMP 4.0 offload (GPUs)

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;
```

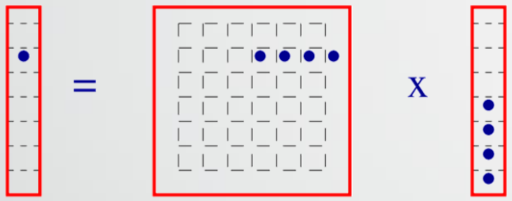
What is the best SIMD strategy?

Example Matrix x Vector multiplication

QCD (3x3 matrices) does not fit very nicely the SIMD vectors

Vector = Matrix x Vector

Natural approach

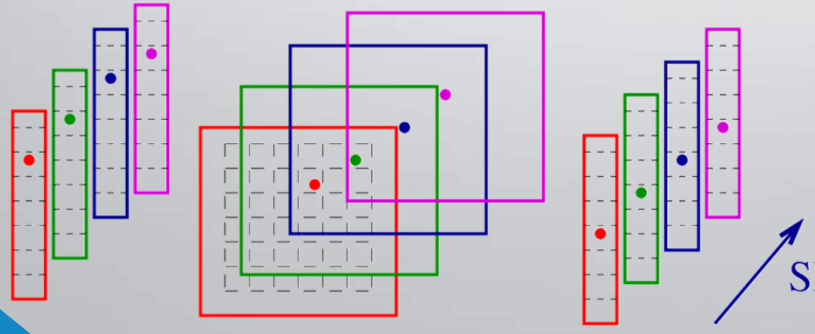


Reduction of vector sum
is bottleneck for small N

```
inline template<int N, class simd>
void matmul( simd *x, simd *y, simd *z)
{
   for(int i=0;i<N;i++){
      for(int j=0;j<N;j++){
         x[i] = x[i]+y[i*N+j]*z[j];
      }
   }
}
```

Grid approach

Many vectors = many matrices x many vectors



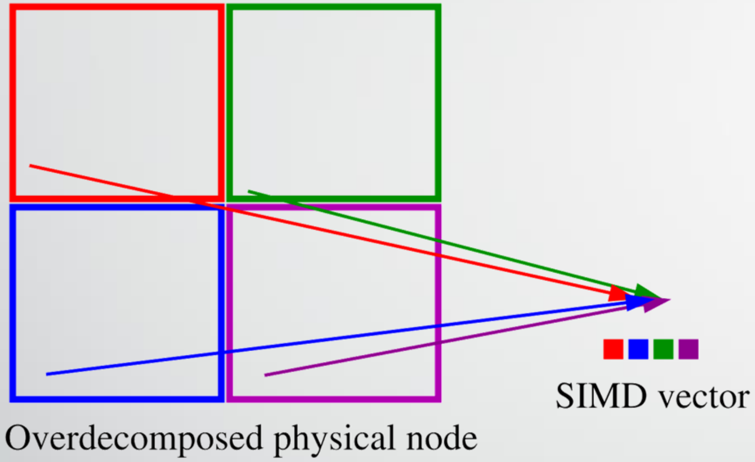
SIMD interleave

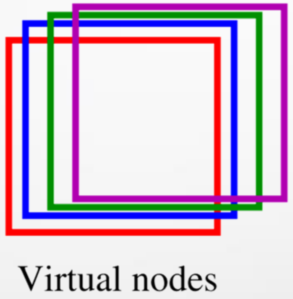
No reduction or SIMD lane crossing operations.

100% SIMD efficiency

Back to Connection Machines

- The SIMD Connection Machines in the '80 had similar problems
- Solution: map the vector units to virtual nodes (**cmfortran** and **HPFortran**)





Virtual nodes layout

ISA	vRealF	vRealD	vComplexF	vComplexD	default layout
SSE	4	2	2	1	1.1.1.2
AVX	8	4	4	2	1.1.2.2
AVX512	16	8	8	4	1.2.2.2

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

Message: OVEDECOMPOSE & INTERLEAVE

Grid data parallel template library - I

- 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

```
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 data parallel template library - II

- Expression templates engine
  - Under 350 lines of code (harnessing C++11 type inference)

```
template<class l,class r,int N> inline
auto operator * (const iMatrix<l,N>& lhs,const iVector<r,N>& rhs)
-> iVector<decltype(lhs._internal[0][0]*rhs._internal[0]),N>
{
  typedef decltype(lhs._internal[0][0]*rhs._internal[0]) ret_t;
  iVector<ret_t,N> ret;
  for(int c1=0;c1<N;c1++){
     mult(&ret._internal[c1],&lhs._internal[c1][0],&rhs._internal[0]);
     for(int c2=1;c2<N;c2++){
        mac(&ret._internal[c1],&lhs._internal[c1][c2],&rhs._internal[c2]);
     }
  }
  return ret;
}
```

- Variadic macros for IO serialisation

Stencil support

Pass the stencil a list of directions and displacements

```
int npoint;
std::vector<int> directions    ;
std::vector<int> displacements;
CartesianStencil Stencil(&CoarseGrid,npoint,Even,directions,displacements)
```

Coarse grid operator in Grid

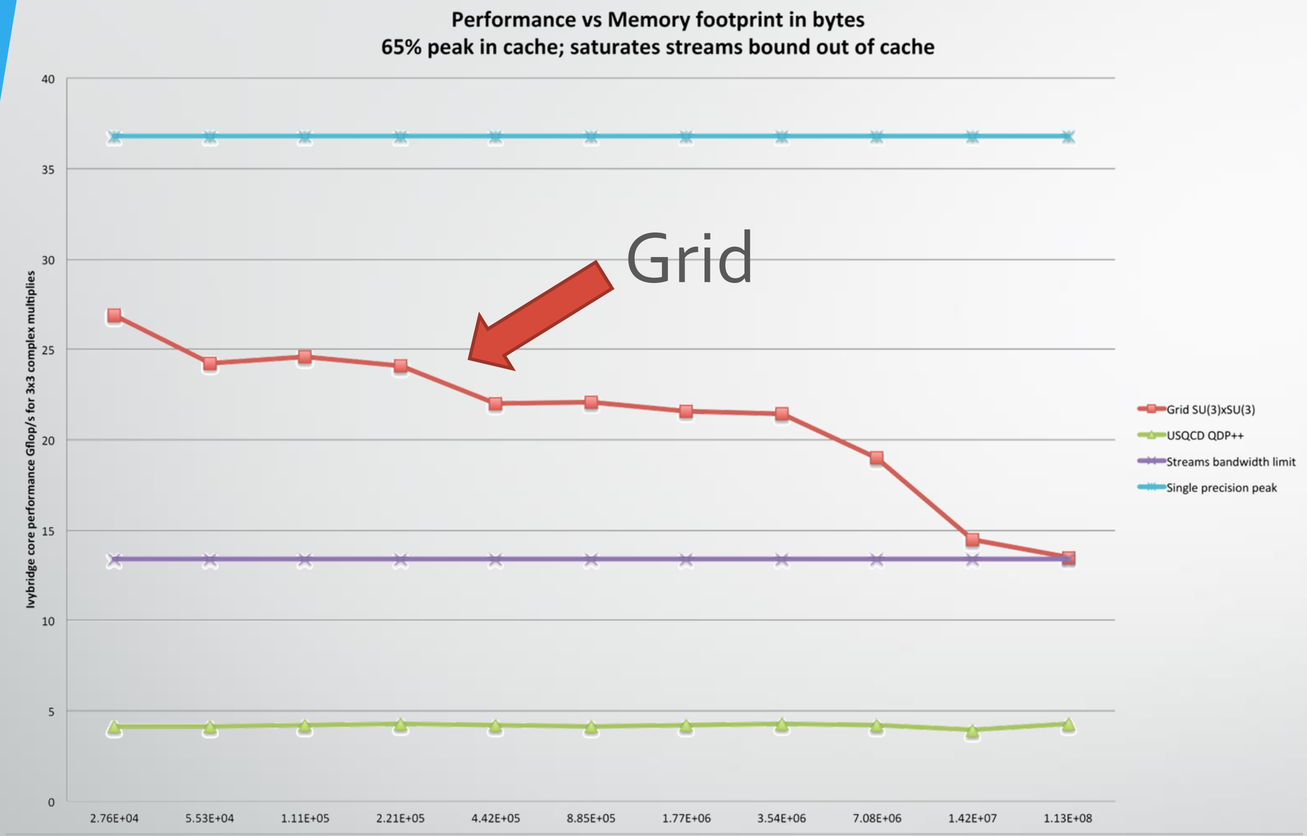
```
void M (const CoarseVector &in, CoarseVector &out){
  conformable(_grid,in._grid);
  conformable(in._grid,out._grid);

  SimpleCompressor<siteVector> compressor;
  Stencil.HaloExchange(in,comm_buf,compressor);

PARALLEL_FOR_LOOP
  for(int ss=0;ss<Grid()->oSites();ss++){
    siteVector res = zero;
    siteVector nbr;
    int offset,local,perm,ptype;

    for(int point=0;point<geom.npoint;point++){
      offset = Stencil._offsets [point][ss];
      local  = Stencil._is_local[point][ss];
      perm   = Stencil._permute [point][ss];
      ptype  = Stencil._permute_type[point];

      if(local&&perm) {
        permute(nbr,in._odata[offset],ptype);
      } else if(local) {
        nbr = in._odata[offset];
      } else {
        nbr = comm_buf[offset];
      }
      res = res + A[point]._odata[ss]*nbr;
    }
    vstream(out._odata[ss],res);
  }
  return norm2(out);
};
```

Stencil organises halo exchange for any vector type; compressor can do spin proj for Wilson fermions.

Stencil provides index of each neighbour (knows the geometry)

User dictates how to treat the internal indices in operator

High level code performance - I




```
std:vector<int> grid ({ 8,8,8,8 });
std:vector<int> simd ({ 1,1,2,2 });
std:vector<int> mpi ({ 1,1,1,1 });
std:vector<int> threads ({ 1,1,1,1 });

CartesianGrid
Grid(grid,threads,simd,mpi);

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

A = B * C;
```

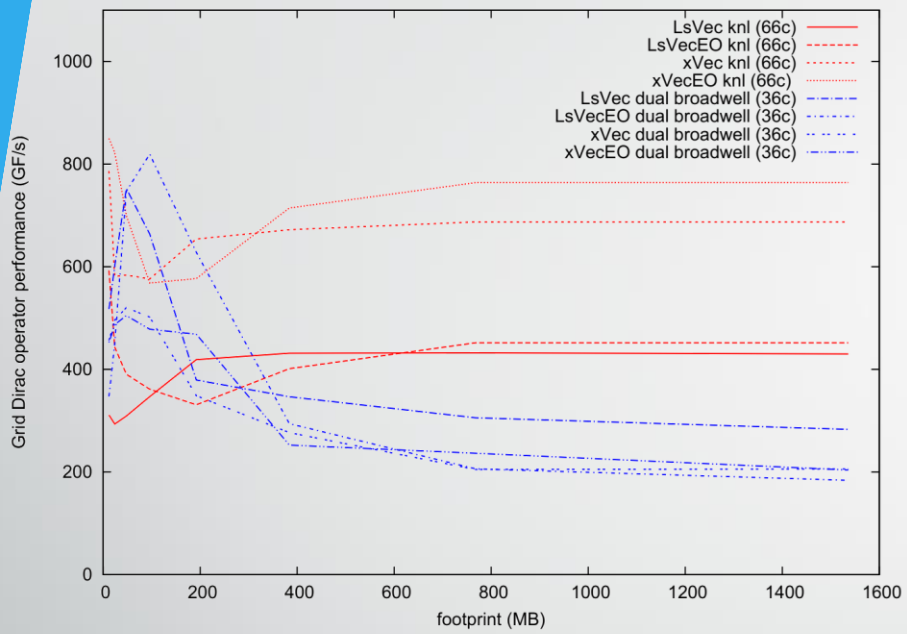
SU(3) matrix multiply on Intel-i7-3615QM (AVX)
Single precision, 65% of peak when in cache

High level code performance - II

Dirac operator D_w application performance

Architecture	Cores	Gflops/s (L_s x D_w)	Peak
Intel Knight's Landing 7250	68	770	6100
Intel Knight's Corner	60	270	2400
Intel Broadwell x2	36	800	2700
Intel Haswell x2	32	640	2400
Intel Ivybridge x2	24	270	920
AMD Interlagos x4	32 (16)	80	628

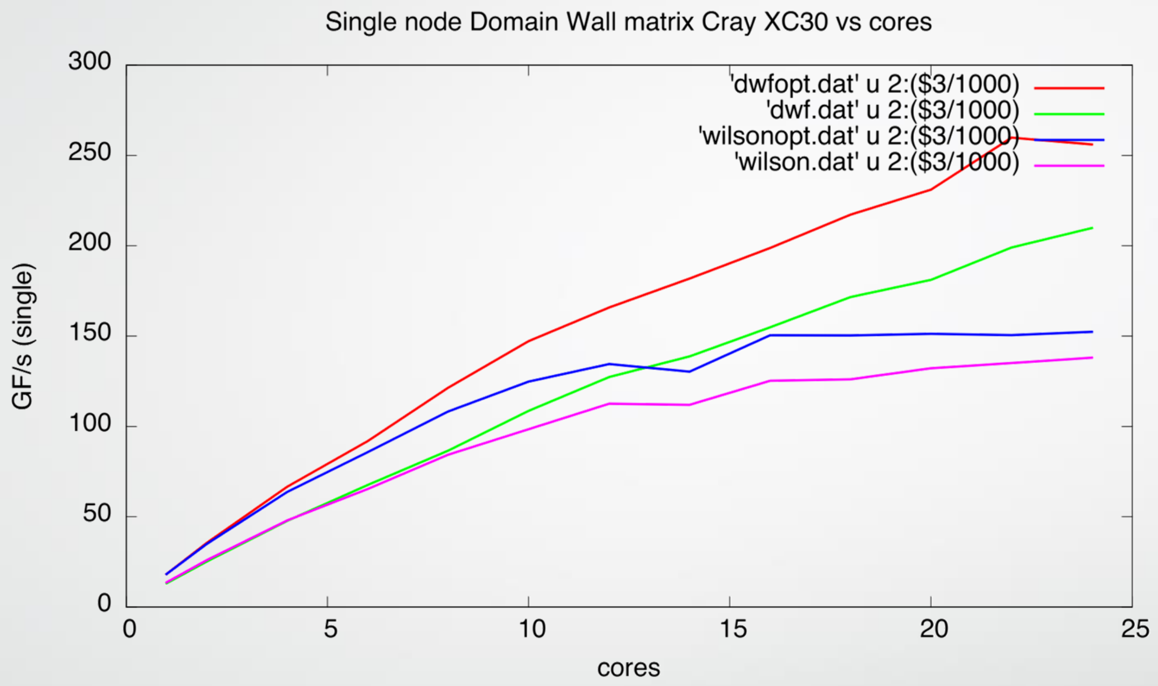
High level code performance - III



- Grid single node, single precision performance for multiRHS Wilson term
- Knight's Landing 7250, 68 core
  - Used 66 cores - a few empty cores usually faster
- One KNL substantially faster than two Broadwell's (18+18) out of cache

- 1 thread per core fastest after writing in assembler (*not* intrinsics)
  - Macro system and mixed C++/asm minimises pain
  - Hand allocation of registers evades stack eviction, cache more deterministic
  - Hand prefetch to L2 and to L1
  - 8.2.2.2 cache blocking
- Single core instructions-per-cycle (IPC) is 1.7 (85% of theoretical, 2 IPC)
- Multi-core L1 hit rate is 99% (perfect SFW prefetching)
- Multi-core MCDRAM bandwidth 97% (370GB/s)

g++-4.9 on Xeon Ivybridge nodes



- $8^4 \times 8$ local volume
- Dual 12 core 2.7 GHz Ivybridge (EPCC Archer - Cray XC30)
- Node peak is 1004 GF in single precision.
- 42% of peak on 1 core
- 26% of peak on 24 cores
- Intel and Clang compilers likely higher

Implementation status

- Basic Grid type system essentially complete
  - General algebra and stencil support
  - QCD types, generic SU(N), arbitrary dimensions
  - Simple to port UKQCD observables code over from QDP++
  - Simple ports from the IroIro++ codebase (KEK)
  - Sum, SliceSum
  - Fast Fourier Transform
  - to do: Gauge fixing
- Algorithms
  - CG, MCR, GCR, VPGCR
  - Chebyshev approx, Remez, Multishift CG
  - Multigrid pCG, pGCR
  - Heatbath
  - HMC, RHMC, multilevel integrators
  - Smeared gauge field actions updates

Implementation status

- Fermion Dirac operators
  - Even-odd and unpreconditioned have a single unified definition
  - Wilson
  - {Wilson, Shamir, Mobius } – Kernel, 5d chiral fermions
  - Periodic and G-parity boundary conditions
- Hadronic measurements
  - Module based framework
  - Minimisation of computing resources (experimental)
    - Tree-network topology based scheduling of complex jobs
- Support non-QCD field theories
  - adjoint-representation complete
  - two index symmetric and antisymmetric representations soon

Final Notes

- GitHub
  - www.github.com/paboyle/Grid
  - Gitflow for workflow management
  - Travis CI for automated testing and deploy
  - Nightly builds
- ISA support:
  - SSE, AVX, AVX2, AVX512, IMCI
  - Neon (ARM), partial
  - QPX (BG/Q)
  - Plan for OpenMP 4.0 offload targets (GPU?)
  - 400 lines of code for implementation of a new architecture

In our (unbiased !?) view it is rather good!

THE UNIVERSITY of EDINBURGH
School of Physics
and Astronomy

Grid

a next generation data parallel C++ library

Peter Boyle, **Guido Cossu**,
Antonin Portelli, Azusa Yamaguchi

Work funded as an Intel Parallel Computing Centre

September 8th 2016, Edinburgh
DiracDay 2016