Grid : Data Parallel QCD Library

Peter A Boyle, Azusa Yamaguchi

Intel Parallel Computing Centre @ Higgs Centre for Theoretical Physics, University of Edinburgh

Guido Cossu

KEK, Tsukuba

We present progress on a new C++ data parallel QCD library. It enables the description of cartesian fields of arbitrary tensor mathematical types.

Ddata parallel interface, conformable array syntax with Cshift and masked operation (c.f. QDP++, cmfortran or HPF).

Three distinct forms of parallelism are transparently used underneath the single simple interface:

- **MPI task parallelism**
- **OpenMP thread parallelism**
- **SIMD vector parallelism.**

The SIMD vector parallelism achieves nearly 100% SIMD efficiency due to the adoption of a virtual node layout transformation, similar to those in the Connection Machine.

This ensures identical and independent work lies in adjacent SIMD lanes. SSE, AVX, AVX2, AVX512 and Arm Neon SIMD targets are implemented.

The library is under development. Solvers for Wilson, Domain, and multiple 5d chiral fermions (Cayley, Continued fraction, partial fraction) are implemented.

Features differing from QDP++:

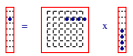
- **Shift by arbitrary distance**
- **Storage:**
  - **Checkerboarded fields are half length**
  - **5d fields are same type as 4d with different Grid.**
- **Multiple grids and Projection/Promotion support**
  - **blockProject, blockPromote, blockInnerProduct**
- **Stencil object**
  - **encapsulates geometry of operation**
  - **Performs halo exchange**
  - **Simple to write kernel for Dirac operators.**
- **C++11 : expression template engine < 200 LoC**

What is the best SIMD strategy?

SIMD most efficient for *independent but identical* work
e.g. apply *N* small dense matrix-vector multiplies in parallel:

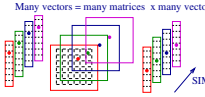
```
inline template<int N, class simd>
void matmul( simd * __restrict__ x,
             simd * __restrict__ y,
             simd * __restrict__ z)
{
  for(int i=0;i<N;i++){
    for(int j=0;j<N;j++){
      fmac(y[i*N+j],z[j],x[i]);
    }
  }
}
```

Vector = Matrix x Vector



Reduction of vector sum is bottleneck for small N

Many vectors = many matrices x many vectors

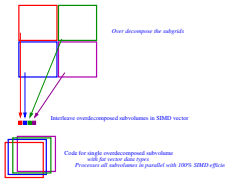


No reduction or SIMD lane crossing operations.

SIMD interleave

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 four virtual nodes



- Conclusion: Modify data layout to align data parallel operations to SIMD hardware
- Conformable array operations simple and vectorise perfectly

vSIMD performance portable SIMD library

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

```
#if defined (AVX1) || defined (AVX2)
    typedef __m256 dvec;
#endif
#if defined (SSE2)
    typedef __m128 dvec;
#endif
#if defined (AVX512)
    typedef __m512 dvec;
#endif
#if defined (QPX)
    typedef vector4double dvec;
#endif
#if defined (GPUSIMD*4)
    typedef double dvec[4];
#endif
    class vRealD  {
        dvec 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(dvec)/sizeof(double);}
    }
```

GRID data parallel template library

Ordering	Layout	Vectorisation	Data Reuse
Microprocessor	Array-of-Structs (AoS)	Hard	Maximised
Vector	Struct-of-Array (SoA)	Easy	Minimised
Bagel	Array-of-structs-of-short-vectors (AoSoSV)	Easy	Maximised

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

```
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];
};

typedef Lattice<iMatrix<ComplexD> > LatticeColourMatrix;
typedef iMatrix<ComplexD>  ColourMatrix;
```

- Define matrix, vector, scalar operations site operations
- Conformable array operations are data parallel on the same Grid layout
- 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*
- *High-level data parallel code gets 160 GF (single) on KNC*
- Single data parallelism model targets BOTH SIMD and threads efficiently.

www.github.com/paboyle/Grid

GRID data parallel CSHIFT details

- Crossing between SIMD lanes restricted to during cshifts between virtual nodes
- Code for *N*-virtual nodes is identical to scalar code for one, except datum is *N* fold bigger

$$\underbrace{(A,B,C,D)}_{\text{virtual subnode}} \quad \underbrace{(E,F,G,H)}_{\text{virtual subnode}} \rightarrow \underbrace{(AE,BF,CG,DH)}_{\text{Packed SIMD}}$$

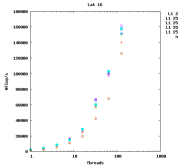
- CSHIFT involves a CSHIFT of SIMD, and a permute *only* on the surface

$$(AE,BF,CG,DH) \rightarrow \underbrace{(BF,CG,DH,AE)}_{\text{cshift bulk}}$$

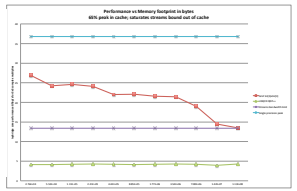
$$\underbrace{(BF,CG,DH,EA)}_{\text{permute face}} \rightarrow \underbrace{(B,C,D,E)}_{\text{virtual subnode}} \quad \underbrace{(F,G,H,A)}_{\text{virtual subnode}}$$

- Shuffle overhead is *suppressed* by surface to volume ratio

High level code performance



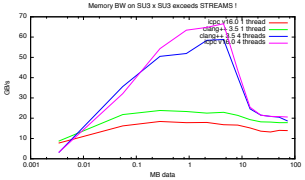
SU3xSU3 XeonPhi

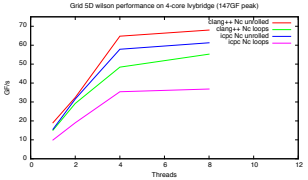


SU3xSU3 AVX

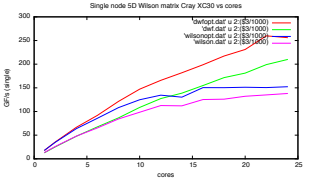
```
std::vector<int> grid ({ 8,8,8,8 });
std::vector<int> simd_layout ({ 1,1,2,2 });
std::vector<int> mpi_layout  ({ 1,1,1,1 });

CartesianGrid Grid(grid,simd_layout,mpi_layout);

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

A = B * C;
```

- Fermion actions
  - Wilson
  - Chiral fermions
    - (Cayley/ContFrac/PartFrac)
    - (Zolotareve/Tanh)
    - (Wilson/Shamir/Mobius)
  - Redblack 4d and unpreconditioned
  - CG, MCR, Multishift CG, GCR
  - Zolotarev, Chebyshev, Remez approx
- Gauge actions
  - Quenched Wilson/Symanzik/Iwasaki
- Multigrid coarse operators
- Nersc file I/O
- Roadmap:
  - (R)HMC, gauge and fermion force terms
  - FFT, measurements
  - Link smearing





Performance on Archer and Edison
Cray XC30 dual Ivybridge 12/24 core.



Stencil support eases the pain of optimised matrix multiplies

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

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

```
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);
};
```