

Computational Methods (practice) - Lecture 1

Peter Boyle (BNL, Edinburgh)

- Introduction & Resources
- Parallel computing and data parallel programming
- Sparse matrices, stencils & PDEs
- Example: (covariant) Laplacian
- Fourier tests
- Gauge invariance tests

Strategy

- Aim to give a discussion of key algorithms and topics in practical Lattice QCD simulation
 - Reasonable degree of rigour, without making this a formal mathematical approach.
 - Provide links to in depth papers and textbooks where appropriate
- *Will illustrate the lectures with practical examples based in Grid*
 - Perhaps a unique feature of this course - let me know what works and does not work !
 - Hope to convince you that lattice code can be small and expressive
- Intersperse discussion of computing/code with discussion of algorithms/physics
 - Hope this is reasonably entertaining
 - Reflects my prejudice that computing is most efficiently learned as required to solve problems
- C++ will be assumed

Assumed programming knowledge

- C++
 - operator overloading
 - template functions and classes - enables generic programming
 - inheritance and object orientation
 - Stroustrup
 - “C” is so much smaller than C++ it is worth learning first: Kernigan and Ritchie
 - <https://www.amazon.co.uk/C-Programming-Language-Bjarne-Stroustrup/dp/0321563840>
- Several keywords in C++, such as “static” are often misnomers, dating to “C” usage

Why C++?

- Low level code is possible;
you can take the compiler out of the way; intrinsics, inline asm etc...
- High level code is possible;
objects and operator overloading allow to define a mathematical type system
Very high level code with natural mathematical expressions
- Single code base can be powerful, portable and efficient
- Broad based compiler support and mature standards process
- CUDA, HIP, SYCL are all C++ based
- Veldhuizen, Expression Templates: *faster than fortran is possible*
- Templating: can write functions & objects (matrices, vectors) of any type multiple precisions,
Real/Complex, matrix of matrix to capture tensorial structures of QFT

How many “abs” functions do we need?

```
#include <math.h>
double fabs(double x);
long double fabsl(long double x);
float fabsf(float x);
                                         template<class t> t abs(t arg) ;

#include <stdlib.h>
int    abs(int i);
long   labs(long i);
long long llabs(long long i);
```

Resources

- Computing:

<https://www.amazon.co.uk/Feynman-Lectures-Computation-Frontiers-Physics/dp/0738202967>

<https://www.amazon.co.uk/Computer-Architecture-Quantitative-Approach-Kaufmann/dp/0128119055>

- Grid:

www.github.com/paboyle/Grid

- Grid documentation:

<https://github.com/paboyle/Grid/raw/develop/documentation/Grid.pdf>

- Examples:

<https://github.com/paboyle/Grid/tree/develop/examples>

- More formal courses
- Tony Kennedy Nara lectures

<https://arxiv.org/abs/hep-lat/0607038>

- Iterative Methods for Sparse Linear Systems, Saad

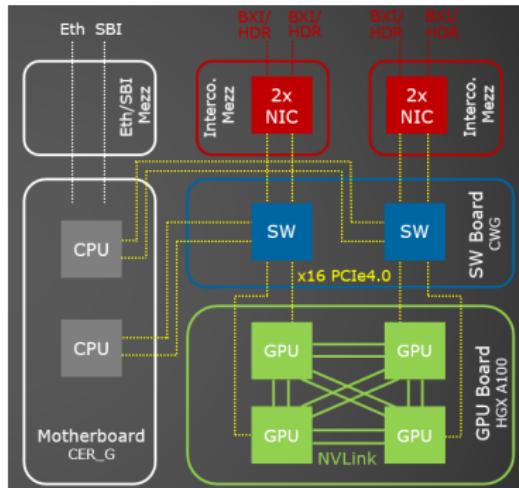
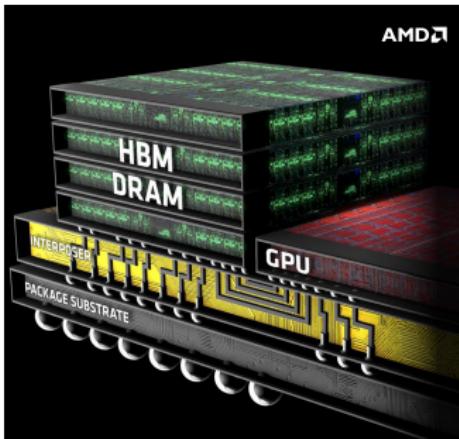
https://www-users.cse.umn.edu/~saad/IterMethBook_2ndEd.pdf

Parallel computing and data parallel programming

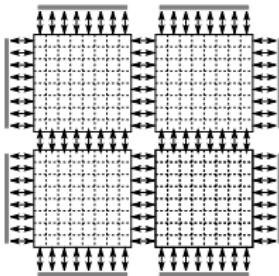
- Comprised of many computing nodes with programmed *message passing* between nodes
- Distributed memory MIMD architecture / loosely coupled multiprocessors
 - Complicated, cooperative access scheme for the data access (send/receive calls)
 - Standardised by: Message Passing Interface (MPI).
 - Predecessors: PVM.
 - Alternatives: UPC, PGAS `Array[node][offset]`
- Nodes themselves contain many layers of parallelism
 - Multiple cores of same type and multiple sockets
 - Accelerators (“vector” cores of different type in a plug-in device)
 - Two or more distinct memory spaces and instruction sets in same programme
 - Multiple vendor supported APIs: CUDA/Nvidia; HIP/AMD; OneAPI/Intel
 - Portability barrier(!)

Parallel computing and data parallel programming

- Modern supercomputers are increasingly difficult to programme
 - This is not the way things used to be, nor in my view the way things should be
- Basic physics dictates that large computers are massively parallel and only locally well interconnected
 - Wires can be long and slow or short and fast (aspect ratio dictates RC delay)
 - Chips must be locally divided into cores
 - “Memory” can be small and fast OR big and slow ⇒ caches
 - 3D “Memory”, HBM and hierarchical memory
 - Interconnect is over a few large high speed wires (or optical transceivers).



Parallel computing and data parallel programming



- 112 nodes of ATOS Sequana XH2000 with 4x A100-40 GPUs per node
- 2x AMD Rome 7H12 CPUs
- 1TB DDR per node
- 4 x HDR-200 Infiniband per node

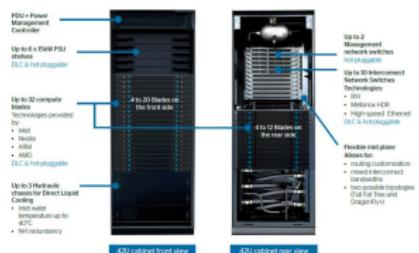


Figure 2-2 BullSequana XH2000 rack

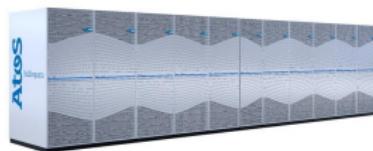


Figure 2-1 BullSequana XH2000 Integrated Supercomputer

Forthcoming Exascale computers



Frontier AMD CPU, AMD GPU; [HIP](#)



Perlmutter AMD CPU, Nvidia GPU; [CUDA](#)



Aurora Intel CPU, Intel GPU; [SYCL](#)

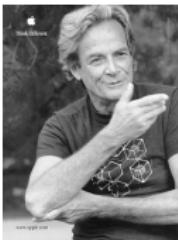


Summit IBM CPU, Nvidia GPU; [CUDA](#)

+ CPU computing is also *not* going away

Parallel computing and data parallel programming

- How to portably target highly parallel hardware ?
 - Easiest if the syntax is high level, and then broken up under an interface
 - Data parallel *used* to mean more than concurrently executed loop bodies – treat entire arrays as objects
 - “Borrow” concepts from the *Connection Machine*



CM-2

- <https://longnow.org/essays/richard-feynman-connection-machine/>
- <https://people.csail.mit.edu/bradley/cm5docs/CMFortranProgrammingGuide.pdf>
- Compare with QDP++
- HPF failed due to poor performance over MPI: **systematic stencil operation assistance**
 - PGAS will probably fail for the same reasons!
- Never break a high level operation into a specific ordering and expect language to reorder Entropy in action! Preserve higher level information

High level data parallelism

- At the highest level deal with a distributed array across a multi-node computer as an aggregate object
 - Periodic CSHIFT / Dirichlet EOSSHIFT primitives
 - Conformable array operations on arrays of identically laid out data
 - c.f. F90 syntax array wildcarding/slicing
 - Predicated data parallel operation (where)
- High Performance Fortran was a failure due to MPI latency
 - Introduce additional, lower-level support for PDE stencil operations & halo exchange.
- Use C++ opaque containers for distributed arrays
 - Operator overloading leaves expressions “natural”
 - (compact C++11, 200 LoC) expression template engine keeps site local expressions optimized.
- Supports CPU w/ SIMD, CUDA, SyCL 2020, HIP compilation targets
 - Device data motion uses either software cache or unified virtual memory

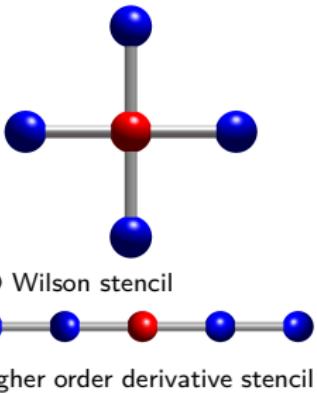
```
A=B*C+Cshift(D,1,Xdir);  
  
face = (mod(coor,Block[mu]) == 0) ;  
  
projected= where(face==1,sp_proj,projected);
```

Sparse matrices, stencils & PDEs

- Encode differential operators as discrete finite difference derivatives
- Think of as sparse matrices - typically band diagonal

$$M(x, y) = \frac{\delta_{x+\mu,y} + \delta_{x-\mu,y} - 2\delta_{x,y}}{a^2}$$

- In typicaly PDE usage, discretisation schemes can be encoded via coefficients and a *stencil*
- *The stencil is a geometric shape encoding the set of points y that contribute to the operator at point x*
- This is translationally invariant and characteristic of a discretisation scheme.
- Naive Laplacian and Wilson Fermions have a $2N_d + 1$ point stencil
- Improved staggered Fermions n discretisations have a $4N_d + 1$ point stencil
- Fourier transformation \Rightarrow phase factor on each term in the stencil defines the lattice Feynman rules
- More non-local actions allow higher order improvement (classical/perturbative/NP)
c.f. Hamber-Wu, Naik terms.



$$\begin{aligned} &c_1(e^{ip_\mu} - e^{-ip_\mu}) \\ &+ c_2(e^{i2p_\mu} - e^{-i2p_\mu}) \\ &\rightarrow ip_\mu + O(p_\mu^5) \end{aligned}$$

Grids primitives (see documentation)

- Common QCD discretisations suggest we can construct most operations with
 - N-dimensional grids
 - Shifting along coordinate axes
 - Gauge covariant of shifting fields
 - Cartesian stencil operators along axes only
 - Generic stencil operators within some taxi-cab distance
 - Site local arithmetic: matrix/vector/scalar addition, subtraction, multiplication, math library functions
 - Reductions: Inner-product, norm2, volume sum, slice summing (N-1 dim)
 - N-dim FFT
 - BlockProjection/Promotion between coarse and fine grids

Example: Free and covariant Laplacian

https://www.github.com/paboyle/Grid/examples/Example_Laplacian.cc

- Base class *defines an interface*.
- Specific derived classes inherit and implement this interface differently
- Algorithms deal with the base class and work with all derived operators

```
////////////////////////////////////////////////////////////////////////
// Grid/algorithms/SparseMatrix.h: Interface defining general sparse matrix, such as a Fermion action
////////////////////////////////////////////////////////////////////////
template<class Field> class SparseMatrixBase {
public:
    virtual GridBase *Grid(void) =0;

    virtual void M      (const Field &in, Field &out)=0;
    virtual void Mdag (const Field &in, Field &out)=0;
    virtual void MdagM(const Field &in, Field &out) {
        Field tmp (in.Grid());
        M(in,tmp);
        Mdag(tmp,out);
    }
};
```

Example: Free Laplacian (low performance)

Introduce free field Cshift based Laplacian:

```
template<class Field> class FreeLaplacianCshift : public SparseMatrixBase<Field>
{
public:
    GridBase *grid;
    FreeLaplacianCshift(GridBase *_grid)
    {
        grid=_grid;
    };
    virtual GridBase *Grid(void) { return grid; };

    virtual void M      (const Field &in, Field &out)
    {
        out = Zero();
        for(int mu=0;mu<Nd-1;mu++) {
            out = out + Cshift(in,mu,1) + Cshift(in,mu,-1) - 2.0 * in;
        }
    };
    virtual void Mdag (const Field &in, Field &out) { M(in,out); } // Laplacian is hermitian
};
```

Example: Covariant Laplacian (low performance)

```
template<class Gimpl, class Field> class CovariantLaplacianCshift : public SparseMatrixBase<Field>
{
    INHERIT_GIMPL_TYPES(Gimpl);

    GridBase *grid;
    GaugeField U;

    CovariantLaplacianCshift(GaugeField &_U) : grid(_U.Grid()), U(_U) { }

    virtual void M (const Field &in, Field &out)
    {
        out=Zero();
        for(int mu=0;mu<Nd-1;mu++) {
            GaugeLinkField Umu = PeekIndex<LorentzIndex>(U, mu); // NB: Inefficent
            out = out + Gimpl::CovShiftForward(Umu,mu,in);
            out = out + Gimpl::CovShiftBackward(Umu,mu,in);
            out = out - 2.0*in;
        }
    };
    virtual void Mdag (const Field &in, Field &out) { M(in,out); } // Laplacian is hermitian
};
```

- The covariant shift is passed in through a policy template parameter class
- Knowledge of boundary conditions is external: we've written Laplacian code that works for periodic *and* charge conjugation boundary conditions
- Wilson loops, Gauge actions, Fermion actions in Grid follow similar design ideas

Fourier test

```
std::cout << " Test A: compare in free Field to \"Feynman rule\" / Convolution theorem " << std::endl;
std::vector<int> dim_mask({1,1,1,0}); // 3d FFT

FFT theFFT(@Grid);

Field out(@Grid), F_out(@Grid), F_in(@Grid);

// FFT the random input vector
theFFT.FFT.dim_mask(F_in,in,dim_mask,FFT::forward);

// Convolution theorem: multiply by Fourier representation of (discrete) Laplacian to apply diff op
LatticeComplexD lap(@Grid); lap = Zero();
LatticeComplexD kmu(@Grid);

for(int mu=0;mu<3;mu++) {
    RealD TwoPiL = M_PI * 2.0/ latt_size[mu];
    LatticeCoordinate(kmu,mu);
    kmu = TwoPiL * kmu;

    // (e^ik_mu + e^-ik_mu - 2) = 2( cos kmu - 1 ) - 2 ( i - k_mu^2/2 - 1 ) *k_mu^2 + O(k^4)
    lap = lap + 2.0*cos(kmu) - 2.0;
}

F_out = lap * F_in;

// Inverse FFT the result
theFFT.FFT.dim_mask(out,F_out,dim_mask,FFT::backward);

std::cout << "Fourier xformed (in) " << norm2(F_in) << std::endl;
std::cout << "Fourier xformed Laplacian x (in) " << norm2(F_out) << std::endl;

std::cout << "Momentum space Laplacian application " << norm2(out) << std::endl;
std::cout << "Stencil Laplacian application " << norm2(out_CLcs) << std::endl;

diff = out_CLcs - out;
std::cout << "diff " << norm2(diff) << std::endl;
```

Output

- Compare to application in momentum space
 - Textbook exercise; use to verify code on unit gauge + a random gauge transform of unit gauge
 - Every Fermion action in Grid knows its own lattice Feynman rule and can *self test* the free propagator
 - In addition to regression to other code bases such as Chroma, Bagel or CPS
 - Add gauge covariance and tests with all Gauge links being highly non-trivial are possible

```
*****
Test A: compare in free Field to "Feynman rule"
*****
Fourier xformed (in)          1.25907e+07
Fourier xformed Laplacian x (in) 5.25498e+08
Momentum space Laplacian application  1.02636e+06
Stencil Laplacian application      1.02636e+06
diff 1.0201e-25
```

Gauge invariance test

```
std::cout << "*****" << std::endl;
std::cout << " Test B: gauge covariance " << std::endl;
std::cout << "*****" << std::endl;

LatticeGaugeField U_GT(&Grid); // Gauge transformed field
LatticeColourMatrix g(&Grid); // local Gauge xform matrix

U_GT = U;
// Make a random xform to teh gauge field
SU<NC>::RandomGaugeTransform(RNG,U_GT,g); // Unit gauge

Field in_GT(&Grid);
Field out_GT(&Grid);

Field out_CLCs_GT(&Grid);
Field out_CLst_GT(&Grid);

CovariantLaplacianShift <PeriodicGimplR,Field> CLCs_GT(U_GT);
CovariantLaplacianStencil<PeriodicGimplR,Field> CLst_GT(U_GT);

in_GT = g*in;
out_GT = g*out_CLCs;

// Check M^* GT_xy in_GT = g(x) M_xy g^-dag(y) g(y) in = g(x) out(x)
CLCs_GT.M(in_GT,out_CLCs_GT);
CLst_GT.M(in_GT,out_CLst_GT);

diff = out_CLCs_GT - out_GT;
std::cout << " Difference between Gauge xformed result and covariant Cahift Laplacian in xformed gauge = " << norm2(diff) << std::endl;

diff = out_CLst_GT - out_GT;
std::cout << " Difference between Gauge xformed result and covariant Stencil Laplacian in xformed gauge = " << norm2(diff) << std::endl;
std::cout << "-----" << std::endl;
```

Output

```
*****
Test B: gauge covariance
*****
Difference between Gauge xformed result and covariant Cshift Laplacian in xformed gauge = 7.28706e-23
Difference between Gauge xformed result and covariant Stencil Laplacian in xformed gauge = 7.28985e-23
```

- Covariance of operators is an reasonably strong check
Many mistakes will break the gauge invariance
- Hadronic correlation functions should be gauge invariant (ignore gauge fixed procedures)
- Variations of this test are a common "test weapon of choice"
A necessary **but not sufficient** condition for correctness, loved by older LQCD scientists
- The above two tests combined - correctness on a random gauge transform of free field prove the operator behaves as intended with non-unit random gauge links on every variable: **a strong check**

Compare different implementations

```
FreeLaplacianCshift<Field> FLcs(&Grid);
FreeLaplacianStencil<Field> FLst(&Grid);

LatticeGaugeField U(&Grid);
SU<NC>::ColdConfiguration(RNG,U);

CovariantLaplacianCshift <PeriodicGimpR,Field> CLcs(U);
CovariantLaplacianStencil<PeriodicGimpR,Field> CLst(U);

std::cout << " Test C: consistency of four different Laplacian implementations " << std::endl;

Field in(&Grid); gaussian(RNG,in);

std::cout << " Input test vector " << norm2(in) << std::endl;

Field out_FLcs(&Grid); Field out_Flst(&Grid);
Field out_CLcs(&Grid); Field out_CLst(&Grid);
Field diff(&Grid);

FLcs.M(in,out_FLcs);
FLst.M(in,out_Flst);
CLcs.M(in,out_CLcs);
CLst.M(in,out_CLst);

std::cout << "-----" << std::endl;
std::cout << " Free cshift output vector " << norm2(out_FLcs) << std::endl;
std::cout << " Free stencil output vector " << norm2(out_Flst) << std::endl;
std::cout << " Cov cshift output vector " << norm2(out_CLcs) << std::endl;
std::cout << " Cov stencil output vector " << norm2(out_CLst) << std::endl;
std::cout << "-----" << std::endl;

diff = out_FLcs - out_Flst;
std::cout << " Difference between free Cshift Laplacian and free Stencil Laplacian = " << norm2(diff) << std::endl;

diff = out_FLcs - out_CLcs;
std::cout << " Difference between free Cshift Laplacian and covariant Cshift Laplacian = " << norm2(diff) << std::endl;

diff = out_FLcs - out_CLst;
std::cout << " Difference between free Cshift Laplacian and covariant Stencil Laplacian = " << norm2(diff) << std::endl;
std::cout << "-----" << std::endl;
```

Output

```
*****
Test C: consistency of four different Laplacian implementations
*****
Input test vector 24591.2
-----
Free cshift    output vector 1.02636e+06
Free stencil   output vector 1.02636e+06
Cov  cshift    output vector 1.02636e+06
Cov  stencil   output vector 1.02636e+06
-----
Difference between free Cshift Laplacian and free Stencil Laplacian      = 2.2926e-26
Difference between free Cshift Laplacian and covariant Cshift Laplacian   = 0
Difference between free Cshift Laplacian and covariant Stencil Laplacian = 2.2926e-26
-----
```

- Stencil vs. Cshift based cross checked.
- Free and covariant implementations on unit gauge
- Can also compare different code bases (Chroma/CPS/Bagel/QUUDA etc...)
 - Weak to a common error (e.g. copy the same Dirac basis?), not scientifically independent
- The Wilson and DWF type actions in Grid have been tested in the above fashion
- Caution: the Clover field strength is identically zero in a transform of free field

Exercise

- Obtain and compile Grid;
- Read and run the example

```
git clone https://www.github.com/paboyle/Grid
cd Grid
./bootstrap.sh
mkdir build
cd build
../configure --enable-simd=GEN --enable-debug
make -j 8
cd examples
./Example_Laplacian
```

- Most x86 laptops will support --enable-simd=AVX2
- Generic vectorisation will work on all systems
- GPU compilation requires many more options