HP-SEE DISSEMINATION & TRAINING
Tirana, 17 November 2011

HMLQCD
Hadron Masses form Lattice QCD

University of Tirana
A.Borici , D.Xhako, R.Zeqirllari

EC FP7 Project HP-SEE

http://www.hp-see.eu/

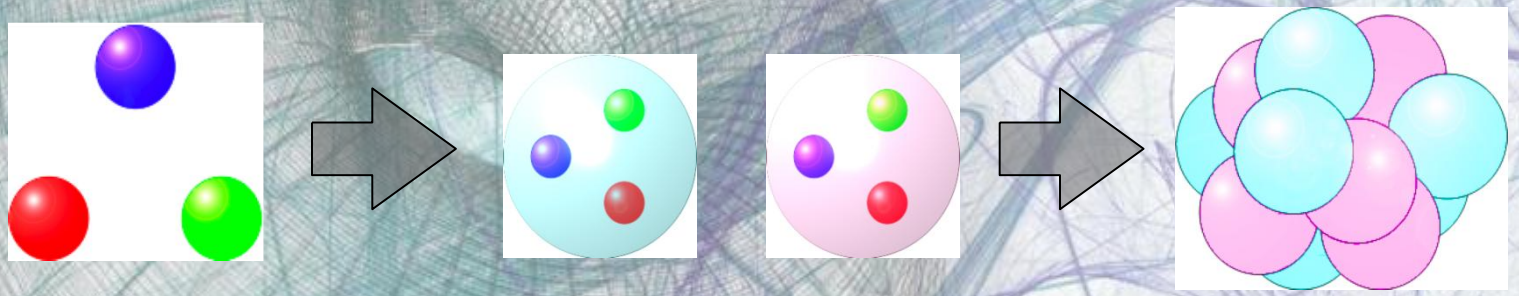
Introduction

Strong interaction one of the fundamental forces in Nature

(gravity, electromagnetic, strong, weak)

dynamics between quarks and gluons

quark proton neutron nucleon

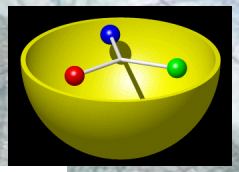


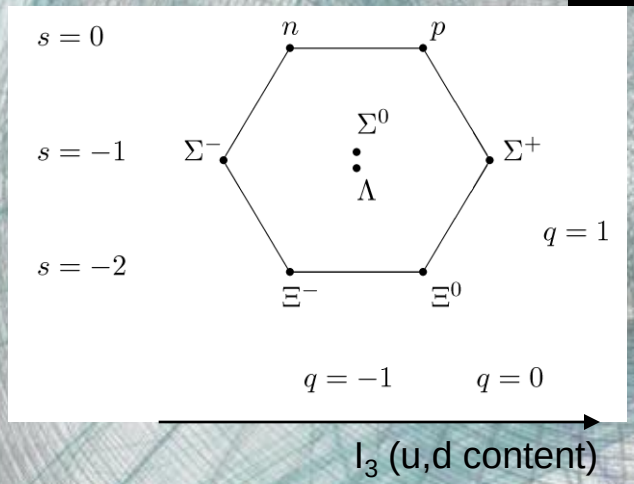
QCD and hadrons

Quarks and gluons are the fundamental particles of QCD
(feature in the Lagrangian)

However, in nature, we observe hadrons:
Color-neutral combinations of quarks, anti-quarks

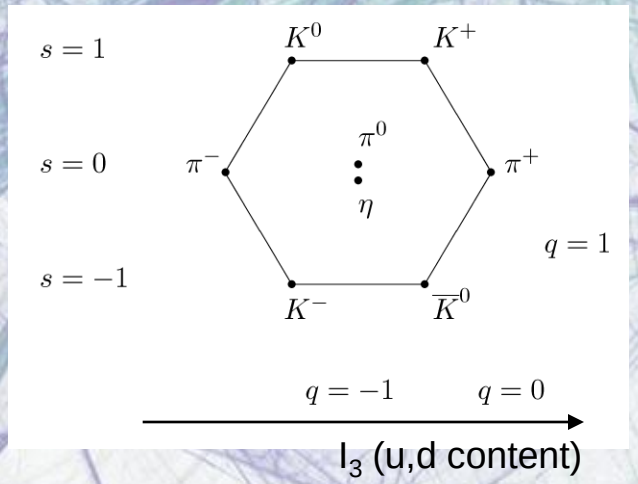
Baryon multiplet





S strangeness

Meson multiplet

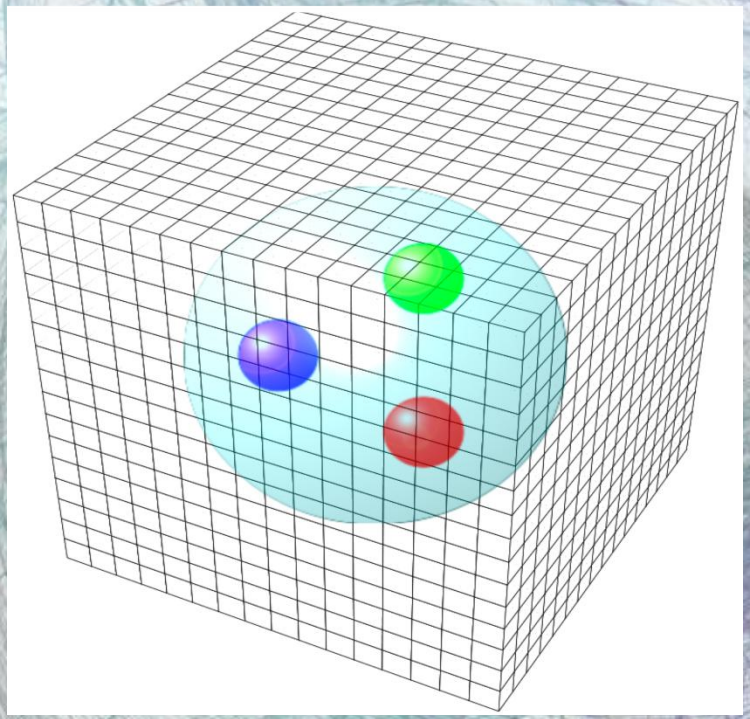


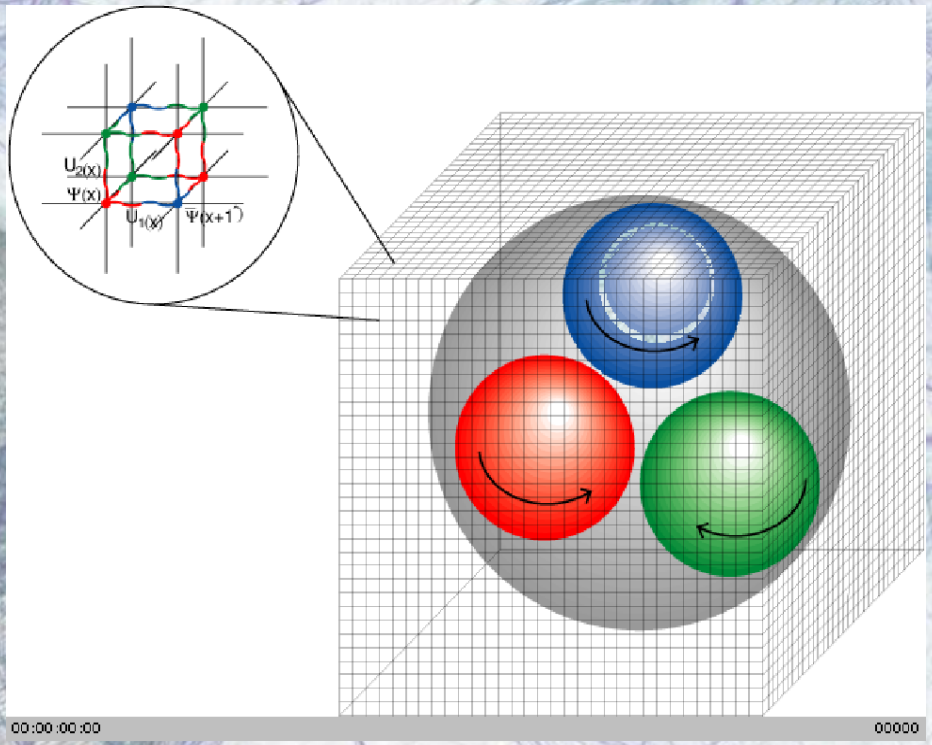
Baryons: 3 quarks

Mesons: quark-anti-quark

Lattice QCD

Nonperturbative investigation of strong interactions with respect to QCD Lagrangian





Numerical simulations of lattice QCD are the only tool to gain information on the hadron masses and scattering phases.

-Lattice gauge theories with fermions are discrete approximations of the theory of strong interactions (QCD).

-Historically, is been impossible to build a chiral fermions theory, without doubled fermions.

- Recently, the idea of minimally doubled fermions, which violate the hypercubic symmetry of the lattice operator, has been revived.

Objectives of HMLQCD

1. Test the hypercubic symmetry breaking effects using Boriçi – Creutz fermions (*Zeqirllari's work*)

2. Compute the mesons masses for large lattices and compare them with the experimental values

3. Simulate Overlap Fermions, with exact chirality, in lattice space. (*Xhako's work*)

4. Develop multigrid algorithm as the fast algorithm to invert a complex operator.

Actually

Using a package software for several calculation in parallel in LQCD, named FermiQCD, specifically:

1. Implementation of Boriçi-Creutz's action (still working with) and calculations in parallel for larger lattices:

   * Calculation of quark propagators

   * Calculation of hadron propagators

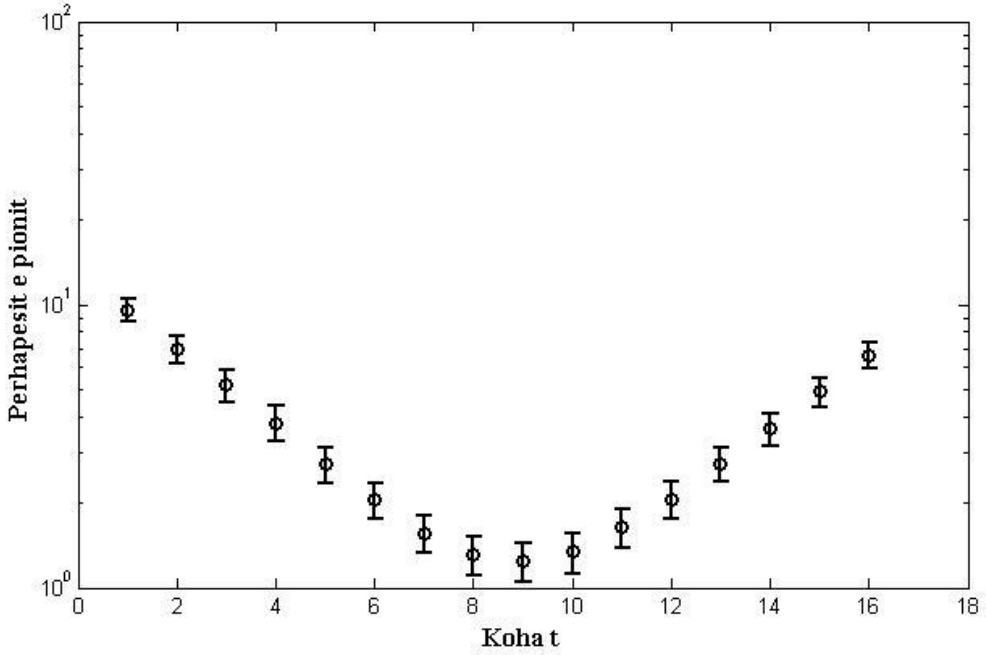
   * Calculation of hadron masses

2. Implementation of Overlap fermions action (still working with) and calculation in parallel for larger lattices:

   * Calculation of quark propagators using multigrid algorithm

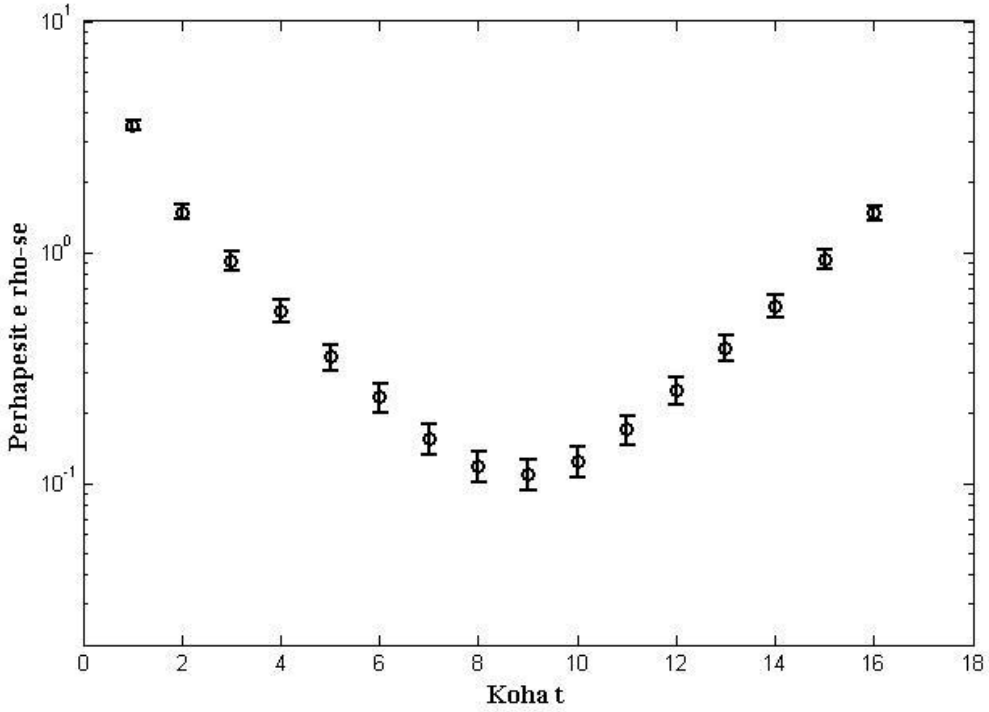
   * Calculation of quark propagators using common algorithm

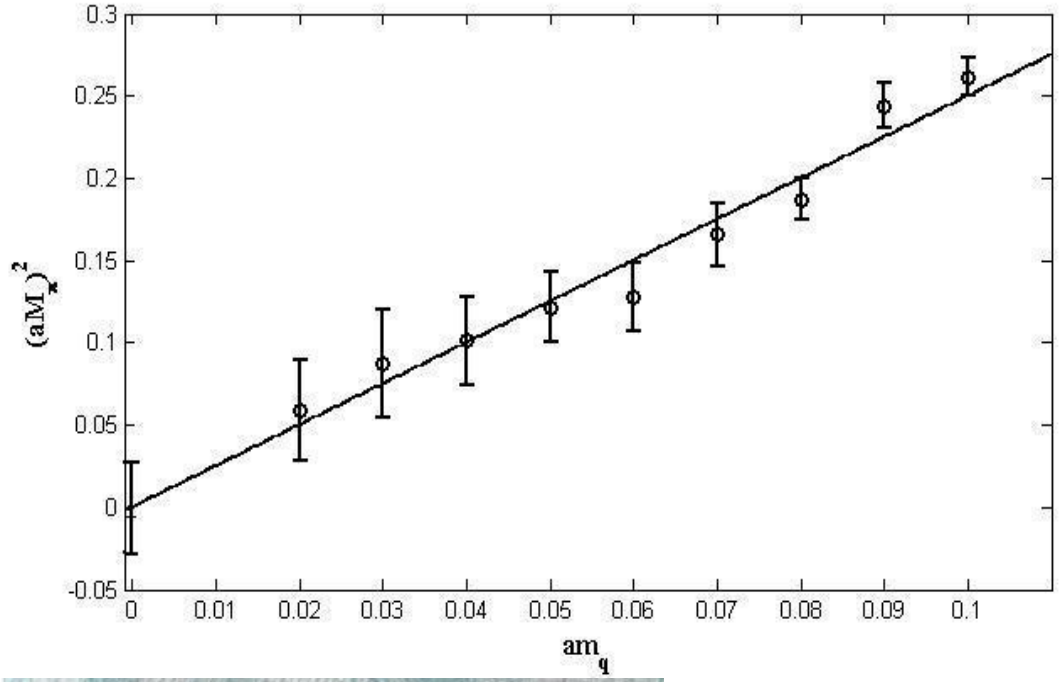
   and compare the results.



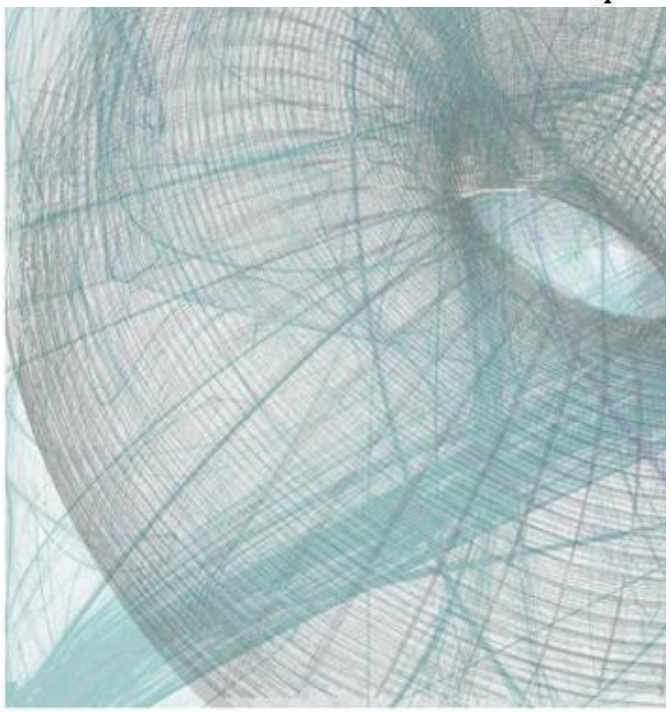
Preliminary results in small lattices

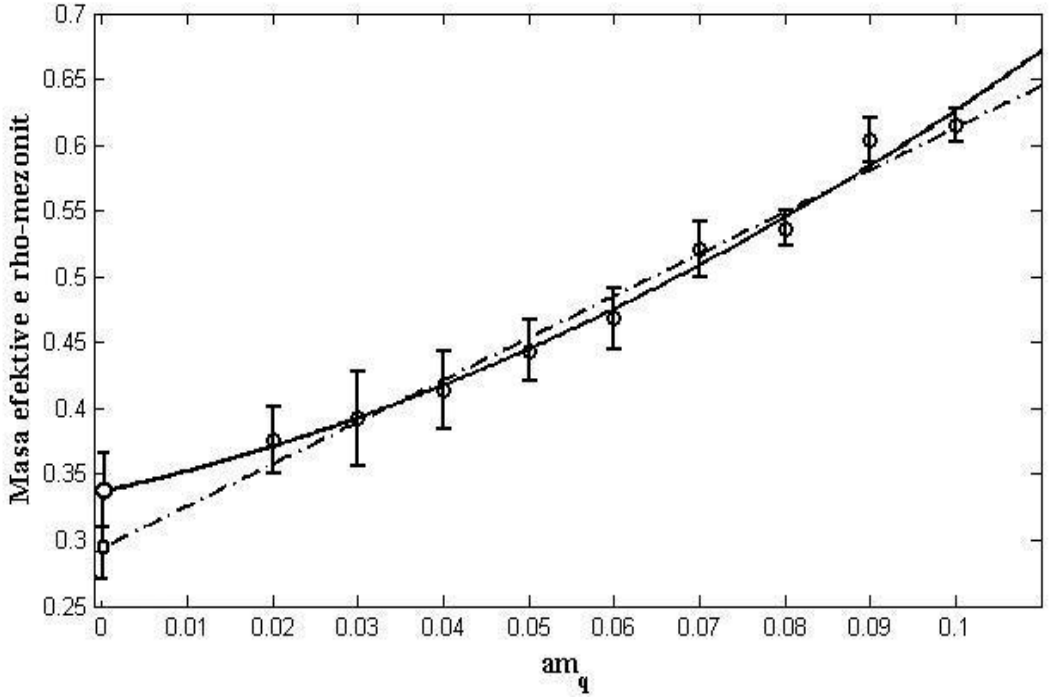
Quark propagators for pion and rho-meson



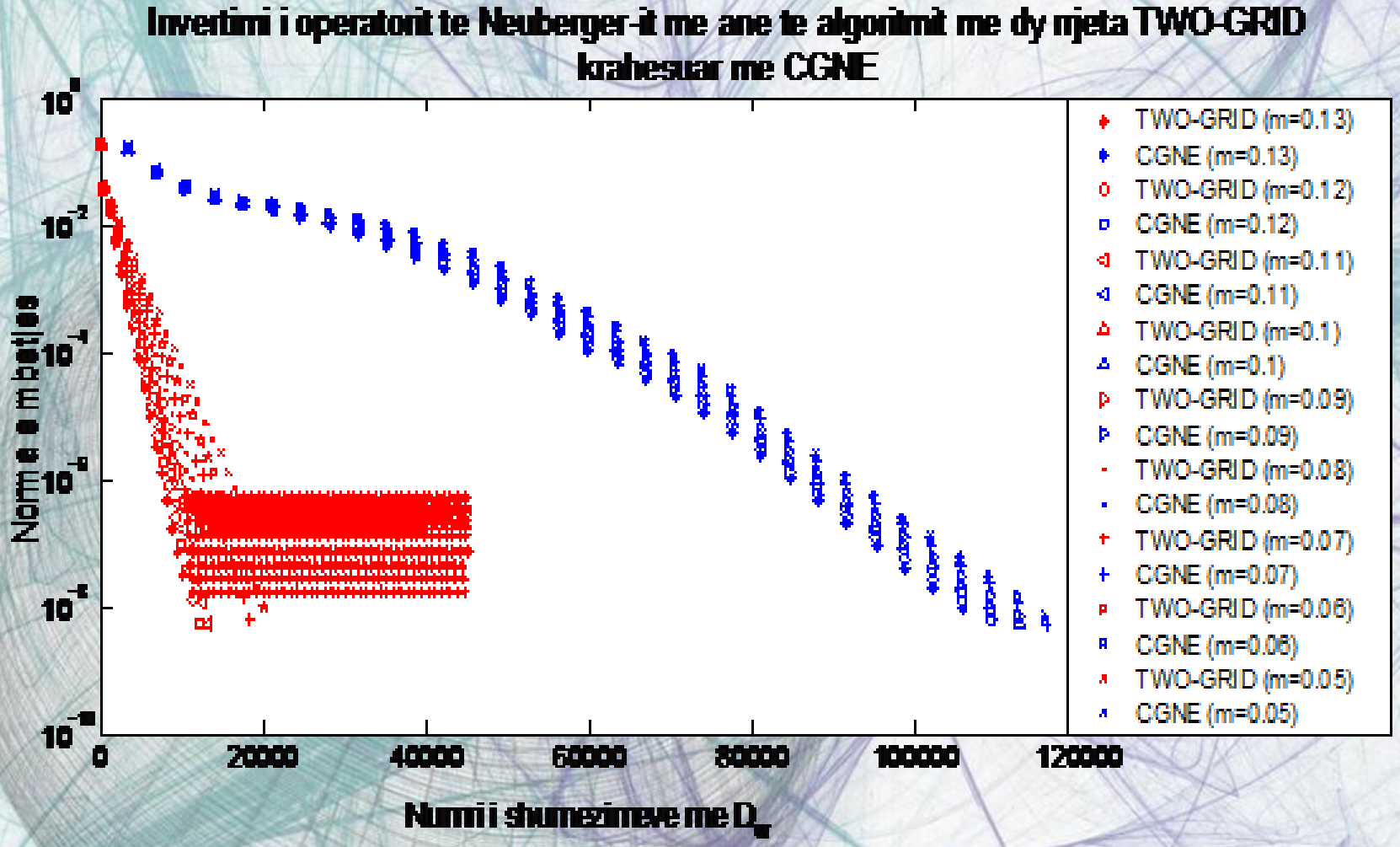


Preliminary results in small lattices



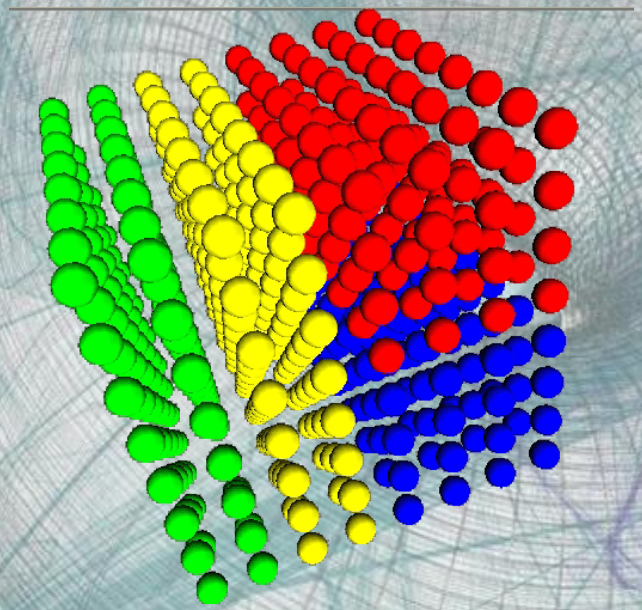


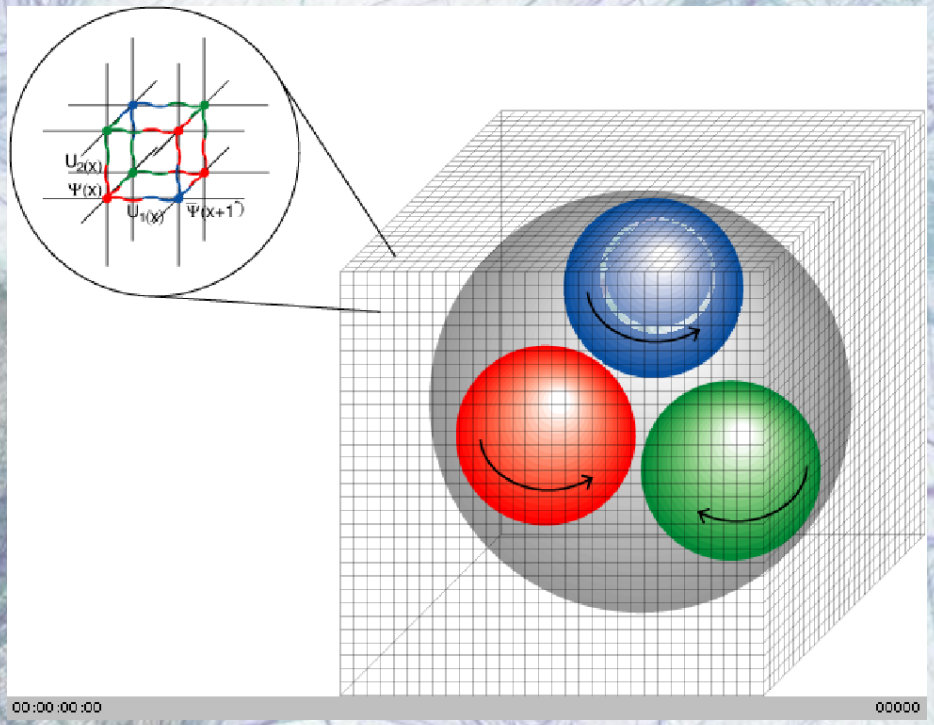
- Preliminary results in small lattices using two_grid algorithm



FermiQCD

```
int box[]={24,8,8,8};
mdp_lattice spacetime(4,box);
fermi_field phi(spacetime,3);
```





Overview

C++ library for fast development of parallel QFT applications

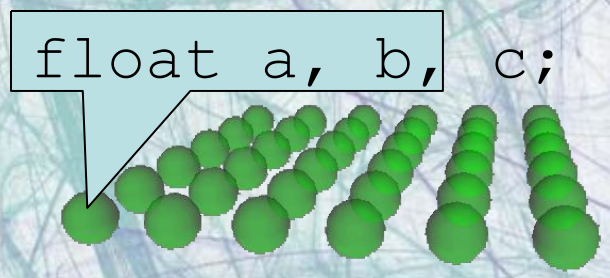
Features:

- fully C++ (uses exceptions, streams, templates, inheritance, etc…)
- top-down design
- includes linear algebra and statistical package
- multiple lattices and fields
- automatic parallelization
- parallel random number generator
- parallel field::**save** and field::**load** methods (inherited)
- **gauge_field** for arbitrary SU(n)
- **fermi_field** for arbitrary SU(n)
- **staggered_field** for arbitrary SU(n) and even ndim
- Wilson, Clover, Asqtad actions (in SSE2 for SU(3)) (un-isotropic)
- Domain Wall action
- Fermilab action for heavy quarks (all dim=6 operators)
- minimum residue, stabilized bi-conjugate and uml inverters
- reads UKQCD, CANOPY, MILC and serial data formats
- easy: no need to know MPI
- safe: no need to use pointers
- flexible: can define your own fields and libraries by inheritance

Linear Algebra

```
mdp_matrix A,B;

A=Random.SU(7);

B=(2+A)*inv(A)*exp(A);

B(0,0)=0.1+0.5*I;

B=(1+Gamma5)*Gamma[2];
```

Gamma matrices




```
struct S {
  float a,b,c;
};

int box[]={6,6};

mdp_lattice space(2,box);

mdp_field<S> phi(space);


forallsites(x)
phi(x).a=space.random(x).plain();
phi.update()

phi.save("myfile.dat");
```

User defined fields

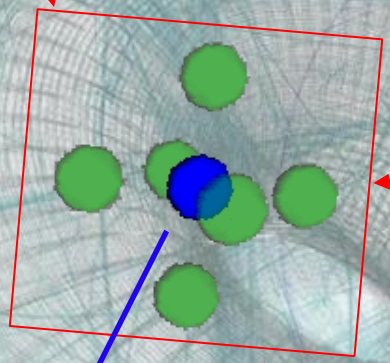
Local random number generat

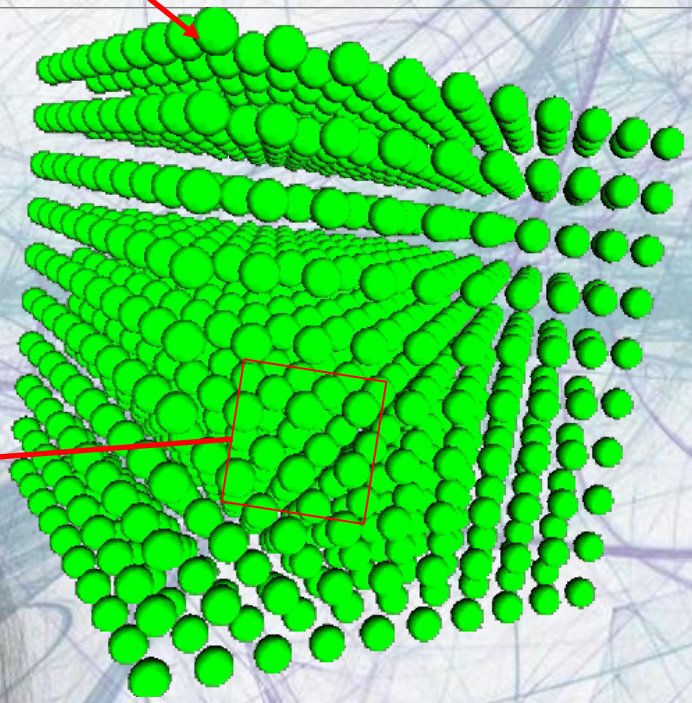
Example

$$\nabla^2\varphi(x) = f(x) \qquad L = \{10, 10, 10\}$$

$$f(x) = A\sin(2\pi x_1 / L_1)$$

$$A = \begin{pmatrix} 1 & i \\ 3 & 1 \end{pmatrix}$$





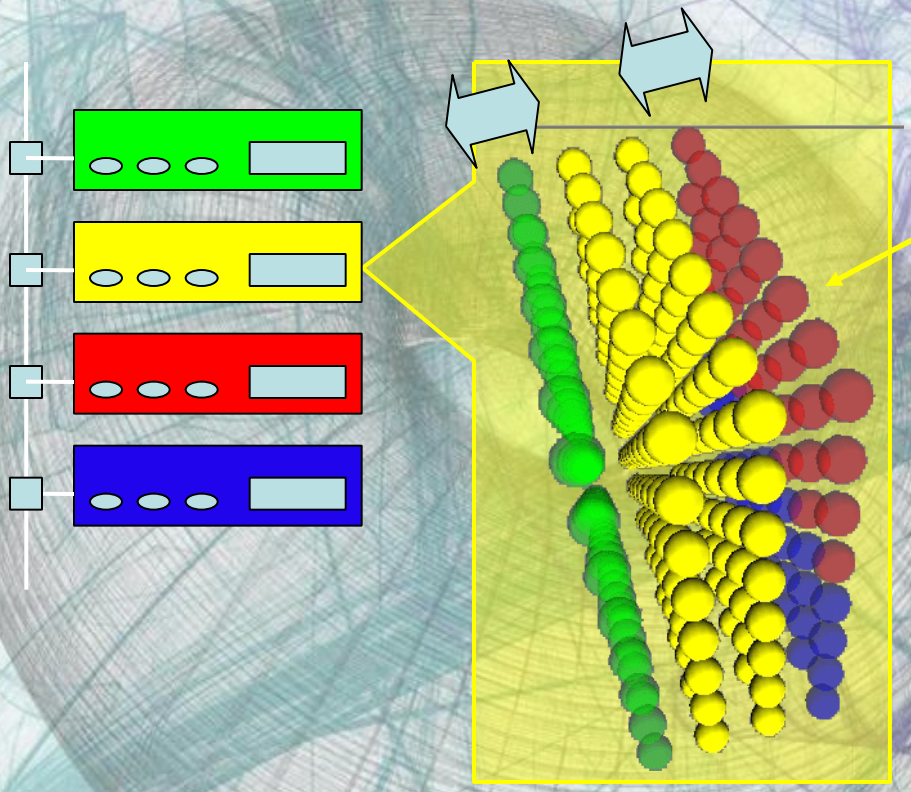
```
phi(x)=(phi(x+0)+phi(x-0)+
        phi(x+1)+phi(x-1)+
        phi(x+2)+phi(x-2)-
        A*sin(2.0*Pi*x(1)/L[1]))/6;
```

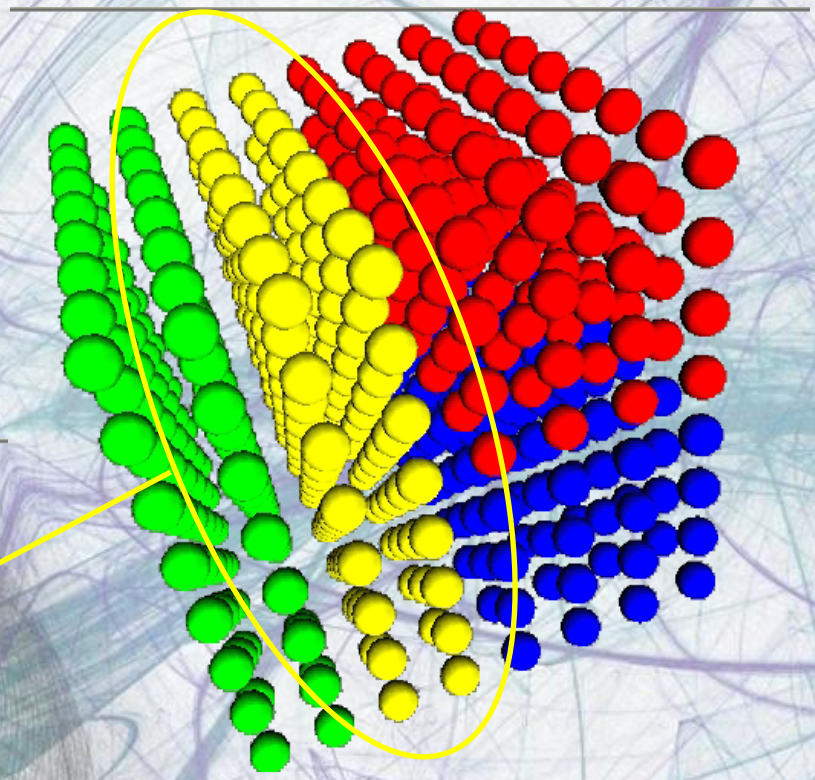
Example: the algorith

```
00    #include "fermiqcd.h"
01
02    void main(int argc, char** argv) {
03       mdp.open_wormholes(argc,argv);     // open communications
04       int L[]={10,10,10};                // declare volume
05       mdp_lattice       space(3,L);      // declare lattice
06       mdp_site          x(space);        // declare site variable
07       mdp_matrix_field phi(space,2,2);   // declare field of 2x2
08       mdp_matrix        A(2,2);          // declare matrix A
09       A(0,0)=1;  A(0,1)=I;
10       A(1,0)=3;  A(1,1)=1;
11
12       forallsites(x)                     // loop (in parallel)
13          phi(x)=0;                       // initialize the field
14       phi.update();                      // communicate!
15
16       for(int i=0; i<1000; i++) {        // iterate 1000 times
17          forallsites(x)                  // loop (in parallel)
18             phi(x)=(phi(x+0)+phi(x-0)+
19                     phi(x+1)+phi(x-1)+
20                     phi(x+2)+phi(x-2)-
21                     A*sin(2.0*Pi*x(1)/L[1]))/6;  // equation
22          phi.update();                  // communicate!
23       }
24       phi.save("field_phi.mdp");         // save field
25       mdp.close_wormholes();             // close communications
26    }
```

Partitioning

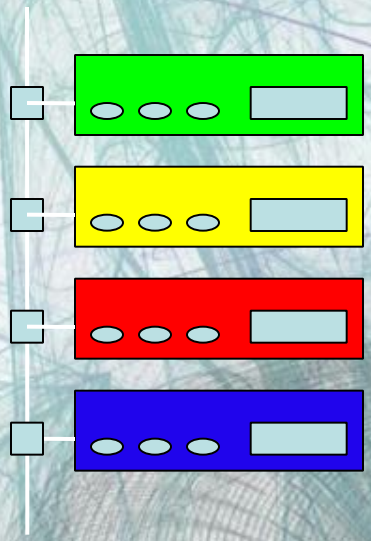
```
int partitioning(int x[]) {
if(x[0]<2) return 0;
if(x[0]<4) return 1;
if(x[1]<4) return 2;
           return 2;
}
```





When a lattice is declared, each field is partitioned, then each site “discovers” its neighbors and each node determines the optimal communication patterns.

Communication Patterns

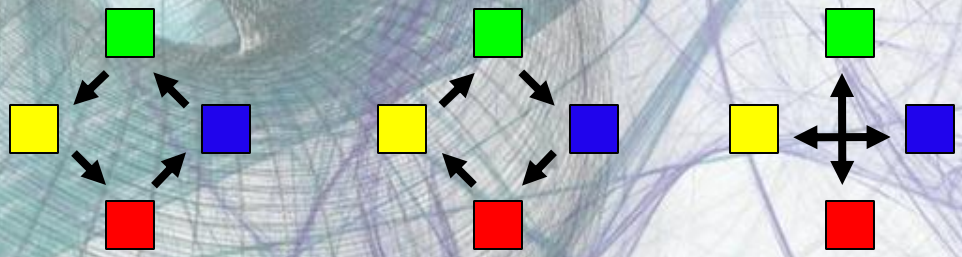


Each node performs communication only when required.

Each node performs one send and one receive at the time.

If two nodes are overlapping, they perform a single send/receive of all site variables.

Only sending nodes use a temporary buffer. The receiving node does not need to reorder the sites.



Parallel average plaquette in 6D S

```
00    #include "fermiqcd.h"

01    void main(int argc, char** argv) {
02      mdp.open_wormholes(argc,argv);     // open communications
03      define_base_matrices("FERMILAB"); // Gamma convention
04     int L[]={24,8,8,8,8,8};          // Lattice volume
05      mdp_lattice       lattice(6,L);     // declare lattice
06      mdp_site          x(lattice);       // declare site variable
07     gauge_field  U(lattice,7);     // declare SU(3) field
08     coefficients gauge;          // declare coefficients

09     gauge["beta"]=4.0;           // set beta
10     set_hot(U);               // set initial gauge

11     for(int i=0; i<1000; i++) {    // loop 1000 times

         // do 10 heatbath steps and print average plaquette!
12       WilsonGaugeAction::heatbath(U,gauge,10);

13       mdp << "<P>=" << average_plaquette(U) << endl;
14       U.save("mygauge.mdp");
15     }
16     mdp.close_wormholes();
17   }
```

Parallel Clover Pion Propagat

```
00    #include "fermiqcd.h"
01    void main(int argc, char** argv) {
02      mdp.open_wormholes(argc,argv);     // open communications
03      define_base_matrices("FERMILAB"); // Gamma convention
04     int L[]={24,8,8,8};                // Lattice volume
05      mdp_lattice       lattice(4,L);    // declare lattice
06      mdp_site          x(lattice);      // declare site variable
07      gauge_field       U(lattice,3);    // declare SU(3) field
08      fermi_propagator S(lattice,3);     // declare propagator
09     coefficients      light_quark; // declare coefficients
10     double        C2[24];       // 2-points correlation func.
11     light_quark["kappa"]=0.1234;  // kappa=0.1234;
12     U.load("mygauge.mdp");              // load a gauge configuration
13       default_fermi_inverter=           // chose action and inverter
           MinimumResidueInverter<fermi_field,gauge_field>;
14       default_fermi_action=FermiCloverActionSSE2::mul_Q;
15     generate(S,light_quark,1e-9,1e-6); // compute propagator on U
17     forallsites(x)                     // contract propagator^2
18       for(int a=0; a<4; a++)
19         for(int b=0; b<4; b++)
20           C2[x(0)]+=real(trace(S(x,a,b)*hermitian(S(x,b,a))));
21     mdp.add(C2,24);                    // parallel sum
22     for(int t=0; t<24; t++)            // print output
23        mdp << t << "\t" << C2[t] << endl;
24     S.save("mypropagator.mdp");    // save propagator
25       mdp.close_wormholes();
26    }
```

Under the hood: Wilson/Clover Ac

```
class FermiCloverActionSlow {
 private:
   // auxiliary functions […]
 public:
  static void mul_Q(fermi_field &psi_out,
           fermi_field &psi_in,
           gauge_field &U,
           coefficients &dict,
           int parity=EVENODD) {
    // declare local variables […]
    // parse action coefficnets […]
    if(dict.has_key("kappa")) kappa=dict["kappa"]; else error();
    // compute Wilson action
    forallsites(x)
      for(mu=0; mu<ndim; mu++) {
        for(a=0; a<nspin; a++) {
          psi_up(a)=U(x,mu)*psi_in(x+mu,a);
          psi_dw(a)=hermitian(U(x-mu,mu))*psi_in(x-mu,a);
        }
        psi_out(x)=psi_in(x)-
          kappa*((1-Gamma[mu])*psi_up+(1+Gamma[mu])*psi_dw);
      }
    // compute Clover term […]
  }
};
```

Under the hood: BiCGSta

```
mul_Q(r,psi_out,U,coeff);
r*=-1;
r+=psi_in;
q=r;  p=0.0;  s=0.0;
rho_old=alpha=omega=1;
do {
      rho=q*r;
      beta=(rho/rho_old)*(alpha/omega);
      rho_old=rho;
      p*=beta;
      p+=r;
      mdp_add_scaled_field(p, -beta*omega, s);
      p.update();
      mul_Q(s,p,U,coeff);
      alpha=rho/(q*s);
      mdp_add_scaled_field(r, -alpha, s);
      r.update();
      mul_Q(t,r,U,coeff);
      omega=t*r;
      omega/= real(t*t);
      mdp_add_scaled_field(psi_out, omega, r);
      mdp_add_scaled_field(psi_out, alpha, p);
      residue=real(r*t);
      residue=sqrt(residue/r.global_size());
      mdp_add_scaled_field(r, -omega, t);
} while (residue>absolute_precision);
```

Same for every action and every field, including user defined.

Next

- Development of the Fermiqcd's codes for Boriçi-Creutz's action.
- Development of the Fermiqcd's codes for Overlap and Domain Wall action.
- Implementation of multigrid algorithm in parallel .
- Increase the lattices, in order to take physical results
- Show how the calculations in parallel improve the performance