Code development (not only) for NSPT

M. Brambilla D. Hesse F. Di Renzo

Università di Parma

Aug 2, 2013

Lattice 2013
XXXI International Symposium on Lattice Field Theory

Outline of the talk

- Motivations: NSPT
- *parmalgt* features
- Parallelization
- Some applications

- A possible strategy for parallelization of the Wilson Dirac operator

NSPT in brief

Parisi and Wu, *Perturbation Theory Without Gauge Fixing* (1981)

- introduce a new degree of freedom *"stochastic time"* $\phi(x) \to \phi(x,t)$
- let the system evolve according Langevin dynamic

$$\frac{\partial \phi(x,t)}{\partial t} = -\frac{\delta S[\phi]}{\delta \phi(x,t)} + \eta(x,t)$$

where $\eta(x,t)$ is a random noise s.t.

$$\langle \eta(x,t) \rangle = 0 \qquad \langle \eta(x,t)\eta(x',t') \rangle = 2\delta(x-x')\delta(t-t')$$

- solve the Langevin equation given some initial condition at $t = t_0$

In the case of gauge theories the resulting Langevin equation is

$$\frac{\partial}{\partial t} U(t) = \left[-i\nabla S[U] - i\eta\right] U(t)$$

Di Renzo, Marchesini, Onofri, *Numerical Stochastic Perturbation Theory* (1993)

- expand the link in power series $U_\mu = \sum_{n=0} \beta^{-n/2} U_\mu^{(n)}$
- plug the expansion in Langevin equation $\Rightarrow$ hierarchy of equations truncated at a given order
- eventually convert differential equations into integral ones

The integration is performed numerically in a perturbative MonteCarlo simulation

One can get rid of the perturbative structure of the computations by defining perturbative operations, e.g.

$$\begin{aligned} A * B &= \left\{ (A*B)^{(0)}, \beta^{-1/2}(A*B)^{(1)}, \beta^{-1}(A*B)^{(2)}, \ldots \right\} \\ (A*B)^{(ord)} &= \sum_{i=0}^{ord} A^{(i)} B^{(ord-i)} \end{aligned}$$

parmalgt

When trying to implement NSPT we had to face some difficulties:

- no simulation software exists that allows a broad flexibility for the data type stored at each lattice site;
- in order to achieve performance LQCD programs are rather difficult to read/modify;
- parallelization is hardly coded in the main routines $\Rightarrow$ trying to modify it can be a pain;

Little history

- TAO language on APE
- Cpp2 ("Enzo's code")
- PRlgt (during my PhD)

(Not) yet another simulation program. . .

parmalgt is a C++ framework which aims to be a general (not specific to QCD) environment where some 'objects' live on a D-dimensional lattice.

We're mainly interested a general environment rather than the "program" itself.

- data type flexibility (C++ templates)
  - scalar
  - gauge (any N)
  - fermions
  - . . .
- any number of physical dimension for free
- perturbative operation already coded (operator overloading)
  - on your own data type symply define basic operations
- hide involved mathematical expressions with standard symbols

Structure of the code

Different layers of code ensure modularity

- Underlying mathematics
  - complex numbers, $N \times N$ matrices, random number generator(s)
  - perturbative structures
  - physical objects: gluon, fermion
- Lattice structure
  - space-time geometry: dimesionality& dimensions, neighbours. . .
  - point, direction: how to move on the lattice
  - field
- Algorithms
  - action
  - gauge update, gauge fixing. . .

Just respect interfaces!

Strategy of algorithms

- main structure: `LocalField<class F, int DIM>`
  - `vector<F>` ⇒ data container
  - `Geometry<DIM>` ⇒ spatial informations
  - `apply_on_timeslice<M>(M& f, const int& t)`
- `IteratorList<D>` pairs behaviour policies ∀ direction
  - `PeriodicPolicy`, `ConstPolicy`, `BulkPolicy`...

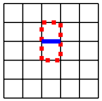
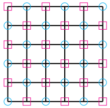
```
template <int D>
struct  TimeSliceIterList  {
  typedef  Pair<PeriodicPolicy,  typename  TimeSliceIterList<D−1>::type > type;
};
```

- `Kernel<Field_t...>` act on a single object of the field
  - void operator()(Field_t& U, const Point& n)
  - checker board hyper cube size

Parallelization

The user can choose between shared memory, distributed memory or a combination of the two

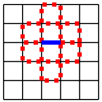
- **multithreading** via OpenMP: for each parallel kernel generate the list of points on which operate. The list is built via a checker-board scheme

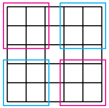



Parallelization

The user can choose between shared memory, distributed memory or a combination of the two

- **multithreading** via OpenMP: for each parallel kernel generate the list of points on which operate. The list is built via a checker-board scheme

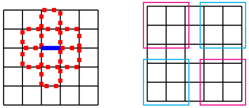




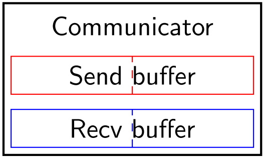
Parallelization

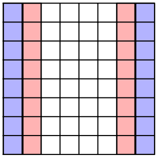
The user can choose between shared memory, distributed memory or a combination of the two

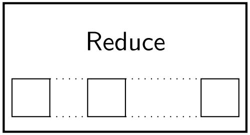
- **multithreading** via OpenMP: for each parallel kernel generate the list of points on which operate. The list is built via a checker-board scheme



- **multiprocessing** (experimental): a `Communicator<Field_t>` contains a copy of the buffer to be sent and a recive buffer. A `Reduce<T>` performs (when required) a reduction of distributed areas.



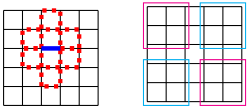




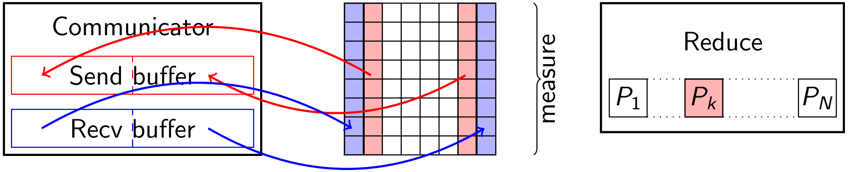
Parallelization

The user can choose between shared memory, distributed memory or a combination of the two

- **multithreading** via OpenMP: for each parallel kernel generate the list of points on which operate. The list is built via a checker-board scheme

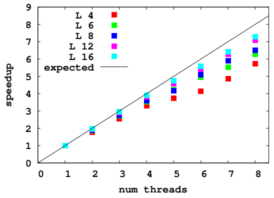


- **multiprocessing** (experimental): a `Communicator<Field_t>` contains a copy of the buffer to be sent and a recive buffer. A `Reduce<T>` performs (when required) a reduction of distributed areas.

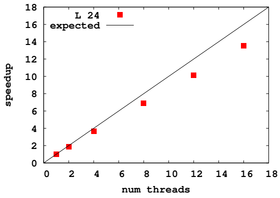


Scaling (multithread only)

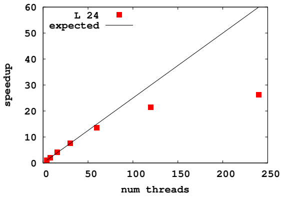
- SF regularization
- Abelian background



(a) Westmere X5680@3.33 GHz



(b) Sandybridge @2.5 GHz



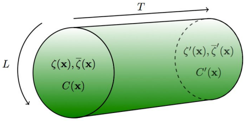
(c) Intel MIC 5110P@1.053 GHz

Multithreading look promising on different architectures, still room for improvement

NSPT for Schrodinger Functional [D.Hesse on Mon]

The boundary conditions

$$U_\mu(x + \hat{k}L) = U_\mu(x)$$
$$U_k(x)\Big|_{x_0=0} = e^{C(x)}$$
$$U_k(x)\Big|_{x_0=T} = e^{C'(x)}$$



T
L
$\zeta(\mathbf{x}), \bar{\zeta}(\mathbf{x})$
$C(\mathbf{x})$
$\zeta'(\mathbf{x}), \bar{\zeta}'(\mathbf{x})$
$C'(\mathbf{x})$

induce a background field that allows to decompose the gauge field (in PT) as

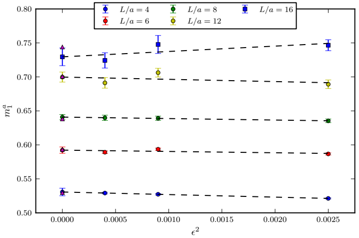
$$U_\mu(x) = \exp\{g_0 q_\mu(x)\} V_\mu(x)$$

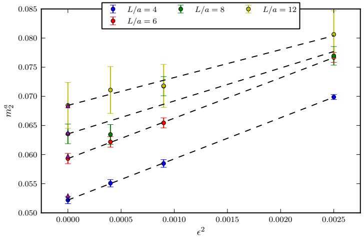
We are dealing with perturbative gluons expanded arond a nontrivial (in this case abelian) background

```
typedef bgf::AbelianBgf Bgf_t; // background field
typedef BGptSU3<Bgf_t, ORD> ptSU3; // group variables
typedef BGptGluon<Bgf_t, ORD, DIM> ptGluon; // gluon
typedef  fields::LocalField<ptGluon, DIM> GluonField;
```

One can define a coupling via

$$g^2_{SF}(L) = \frac{k}{\partial_\eta \ln \mathcal{Z}|_{\eta=0}} = g_0^2 + m_1(L/a)g_0^4 + m_2(L/a)g_0^6 + \dots$$





To give an idea of computational effort consider $L/a = 12$

$$m_2^a = 0.0684(49)$$

with a wall clock time of 210h on a single node of TCD's Lonsdale cluster (AMD Opteron, 8 cores / node)

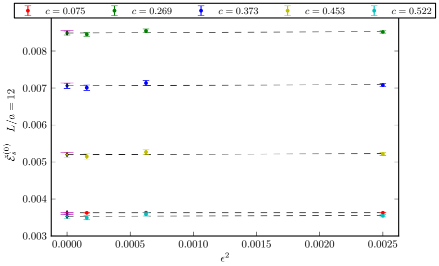
Wilson Flow D.Hesse,M.Dalla Brida [M.Dalla Brida on Mon]

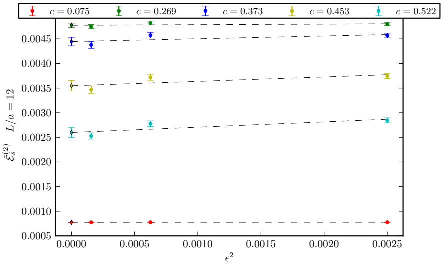
Gradient Flow on the lattice

$$\partial_t V(t)_{x\mu} = -\left\{g_0^2 \nabla_{x\mu} S_G(V(t))\right\} V(t)_{x\mu} \quad V(t)_{x\mu}|_{t=0} = U_{x\mu}$$

One can define a family of running couplings

$$\begin{aligned} \bar{g}^2_{GF}(L) &= \mathcal{N}^{-1}\langle t^2 \hat{E}(t, T/2)\rangle|_{t=c^2L^2/8} \\ &= \check{\mathcal{E}}^{(0)} g_0^2 + \check{\mathcal{E}}^{(1)} g_0^4 + \check{\mathcal{E}}^{(2)} g_0^6 + \cdots \end{aligned}$$
(*P.Fritzsch,A.Ramos*'13)





Conclusions

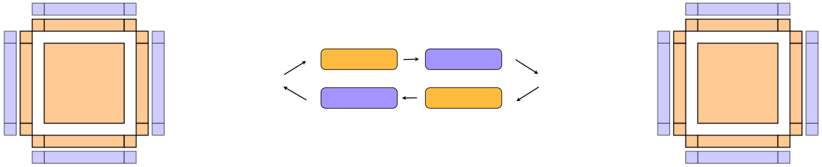
Something has been done

- ✓ periodic/SF boundary conditions
- ✓ trivial/abelian background
- ✓ Langevin gauge update algorithms: euler, 2^{nd} order RK
- ✓ Wilson flow
- ✓ multithreading

and something's on the way

- ✗ multiprocessing
- ✗ fermions
- ✗ action improvements

A strategy for the parallelization of the Wilson Dirac operator

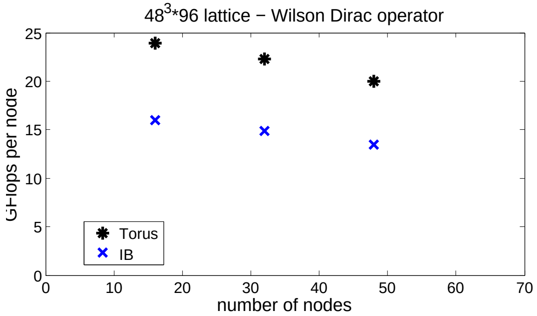


$$\psi'_{\alpha i}(x) = \sum_{\mu=1}^{4} (U_\mu^{ij}(x)(1+\gamma_\mu^{\alpha\beta})\psi_{\beta j}(x+\hat{\mu}) + U_\mu^{ij}(x-\hat{\mu})(1-\gamma_\mu^{\alpha\beta})\psi_{\beta j}(x-\hat{\mu}))$$

- Prepare borders and store them into dedicated buffers
- Half of the threads update the bulk, the remaining half exchange borders (HT: no competition on FP resources!)
- Notice that bulk is actually a bit non-trivial as a concept (one can re-allocate threads, if needed)
- Reconstruct border contributions

Results on Aurora

Wilson Dirac Operator: MPI/IB and atn/TORUS Di Renzo, Lattice2012



Peak performance percentage comparable with ETMC code on the BG/P (but with bigger node granularity).