PROCEEDINGS OF SCIENCE

Code development (not only) for NSPT

Michele Brambilla[*,a], **Dirk Hesse and Francesco Di Renzo**
Università di Parma and INFN, Viale Usberti 7/A, I-43124 Parma, Italy
[a]*E-mail:* michele.brambilla@pr.infn.it,

In recent years NSPT was proven capable of providing high order perturbative results more easily than traditional approaches. The technique is based on numerical integration of the equations of stochastic quantization; one thus gets perturbative results in a way which is alternative to standard Feynman diagrams. One actually needs to solve a hierarchy of stochastic differential equations, the solution being obtained as a perturbative expansion. The key point for an efficient solution is a framework in which everything is computed order by order. We present "parmalgt", a general C++ framework mainly intended for NSPT computations that we are currently developing.

31st International Symposium on Lattice Field Theory - LATTICE 2013
July 29 - August 3, 2013
Mainz, Germany

*Speaker.

PoS(LATTICE 2013)418

1. Motivations

Trying to modify a high performance lattice QCD code can be a hard task. One usually has to go through a hundreds of files usually written with the aim of a high performance and tuned for a few, particular machines, with the side effect of a loss of readability. Things can be even worse if one has to define one's own data type: function calls contain explicit reference to well defined types and cannot be modified easily. Moreover the parallelization strategy usually deeply depends on the geometry of the system one simulates and on the routines of the program. A usual choice is to embed function calls in charge to perform communications in the computational body.
As a result many collaborations develop their own codes, in particular if these are dedicated to some specific application. This is the case of the Parma group and Numerical Stochastic Perturbation Theory.

2. NSPT in brief

In 1981 Parisi and Wu introduced Stochastic Quantization [1] as an alternative to canonical quantization. In this approach one first introduces a new degree of freedom (*stochastic time*), then forces the system to evolve according to the Langevin dynamic

$$\frac{\partial\phi(x,t)}{\partial t} = -\frac{\delta S[\phi]}{\delta\phi(x,t)} + \eta(x,t) \tag{2.1}$$

and finally solves the Langevin equation given some initial condition at $t = t_0$. In (2.1) $\eta(x,t)$ is a gaussian noise s.t.

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

In the case of a gauge theory the Langevin equation (2.1) reads

$$\frac{\partial}{\partial t}U(t) = [-i\nabla S[U] - i\eta]\,U(t), \tag{2.2}$$

where $\nabla = T^a\nabla_a$ is the Lie derivative, defined by $f(e^{i\alpha_a T^a}U) = f(U) + \alpha^a\nabla_a f(U) + \ldots$.

Considering the lattice regularization of gauge theories and expanding the links in a power series $U_\mu = \sum_{n=0}\beta^{-n/2}U_\mu^{(n)}$ Di Renzo, Marchesini and Onofri proposed Numerical Stochastic Perturbation Theory [2]. In this approach one plugs the links' expansions in the Langevin equation (2.2) obtaining a hierarchy of equations to be truncated at a given order. The differential equations can be cast into integral ones and finally integrated numerically on a computer.

The main idea for the numerical approach is to get rid of the perturbative structure of computations defining perturbative operations, e.g.

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

3. *parmalgt*

Because of its peculiarity NSPT has a longstanding story of dedicated programs. The first implementation was written in the TAO language for the apeNEXT machine. In this version lattice size and parallelization strategy were deeply coded in the program.
In order to run simulations on different machines the program was completely rewritten under the name of *Cpp2* (also known as *"Enzo's code"*). *Cpp2* was written, as the name suggest, in C++ whose features like classes and templates allow a broad data type, lattice size and perturbative order flexibility. Parallelization was implemented along one dimension via MPI. Drawback of the highly templated structure is that the program turns out to be difficult to read and modify.
In order to overcome some of the limitations of *Cpp2* we wrote an entirely new program: *PRlgt*. *PRlgt* aims to be a modular C++ program in which lattice structure, data type and algorithms are not deeply connected. This allows to modify some aspects of the code without affecting the rest. In principle it supports any number of physical dimensions, however the user has to encode neighborhood information between the points. Being developed for modern commodity hardware it is designed for multithreading, though multiprocess is not supported.
When we set out to implement the Schrödinger functional [3] in the framework of NSPT we realized some of the limitations of *PRlgt*, in particular perturbative expansion around a nontrivial vacuum requires deep refactoring. Trying to overcome these limitations and exploiting the new standard of C++ (namely c++11) a new simulation environment, *parmalgt*, has been developed. It is not only a LQCD environment, but a general framework where some "objects" live on a D-dimensional lattice.

4. Structure of the code

parmalgt pushes forward some of the aspects we learned from previous implementations in order to obtain most of the benefits. It consists of three main layers:

- a low lying mathematical layer, where basic types and related operations are defined;
- the lattice structure, consisting of the space-time geometry and the abstract concept of a field;
- the algorithms which define the update of the fields and the measurements.

Such a structure allows the data flexibility we are interested in: one starts by defining basic objects like `vector<N>` and matrices `SU<N>` with arbitrary N. Templates and specialization allow to reduce the number of defined objects while ensuring high flexibility. Objects beyond basic types are built using the latter as building blocks: this is the case of perturbative expansions. By means of operator overloading the same operation between different data types is implemented by the same mathematical symbol and resolved by the compiler.
The lattice structure is responsible for the space-time dimensionality `DIM`, the extent of each dimension and the neighborhood relations. Concepts of `Point<DIM>` and `Direction<DIM>` permit to easily reach any neighbor or position in the lattice. Neighborhood relations are recursively computed at compile time by specifying how to sweep every direction. In each direction one can have periodicity (`PeriodicPolicy`), avoid boundaries (`BulkPolicy`) or consider a single slice

(`ConstPolicy`). The basic object required by any simulation is a `LocalField<data_t, DIM>`, which is an abstract container of some kind of data, capable to apply some function called *kernel* on the data it contains.
The algorithmic part of the code is based on the action of the kernel on a single object. Kernels are aware of the type of data on which they act and capable to access all the relevant information when invoked by a field. They can consists of update actions on the object or measurements.

5. Parallelization

The parallelization of the code allows the user to choose among shared memory (multithreading via OpenMP), distributed memory (via MPI) or a combination of the two.
When implementing multithreading one has to take care of data races. Let us consider the case of a next-neighbors action (figure 1(a)). While a thread updates the blue link the red ones must stay the same. A viable approach is the following: for each kernel one generates once and for all the list of points on which to operate, in this case using a checkerboard scheme (figure 1(b)). First of all the threads update the links lying on red points, and only when this update finishes threads start updating the blue ones. This scheme requires only one synchronization for each update of the whole lattice. The approach should be modified when considering updates that require links be-

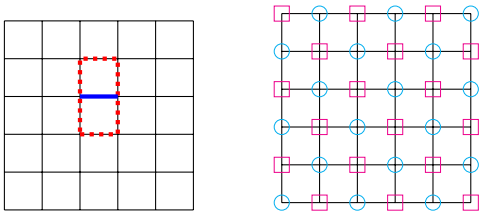


Figure 1: *In case of a multithread next-neighbors update a simple checkerboard scheme is implemented.*

yond next-neighbor (e.g. improved actions, figure 2(a)). In this case the lattice is first divided into "blocks" and subsequently the checkerboard is implemented on the blocks. Referring to figure 2(b) one fills different lists with all the blue points "a", then blue "b" and so on, then switching to red blocks.

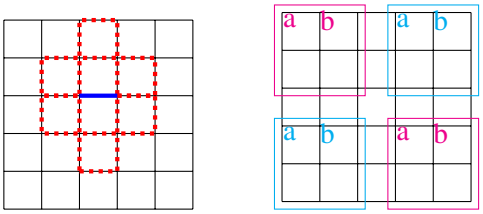


Figure 2: *If the update requires links beyond next-neighbor the multithread uses a blocked-checkerboard scheme.*

Multiprocess parallelization is still experimental. In this approach a `Communicator<Field_t>` is in charge of all tasks required for communications. The lattice can be considered to be composed

of three regions (figure 3): a bulk (white) not dealing with communications, inner boundaries (red) to be updated and sent to neighbors and outer boundaries (blue) received from other processes not to be updated from the current process. The `Communicator` contains a buffer for each region to be sent or received: the first is updated by the current process, copied into the buffer and sent; the latter is received and copied. Operations of send, receive and copy are performed by the `Communicator` itself. An object called `Reduce<T>` takes care of gathering the results of measurements that are performed by each process individually and need to be combined.

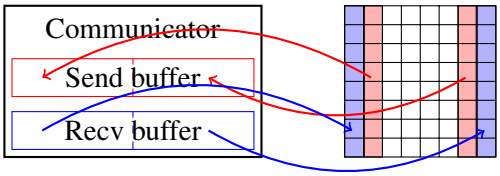
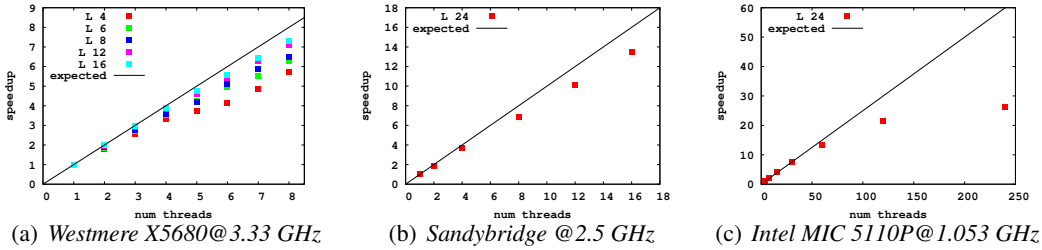


Figure 3: *The* `Communicator` *takes care of communications, by copying data into dedicated areas and performs send/receive operations.*

Scaling tests of the multithread version have been performed for different architectures and different lattice sizes. Figure 4(a) shows the speedup on a dual Intel Westmere X5680@3.33 GHz processor (6 physical cores per processor). Running with 8 threads on a 4^4 lattice gives a speedup of roughly 5.5 (70% theoretical peak), while the speedup on a 16^4 lattice goes up to 7.3 (91% theoretical peak). Figure 4(b) shows the case of a 24^4 lattice on a dual Intel Sandybridge@2.5 GHz (each processor has 8 physical cores). In this case the speedup using 16 threads is 13.5, corresponding to 84% theoretical. On a Intel MIC we are able to reach a limit case of 240 threads on a 24^4 lattice (figure 4(c)). In the latter it is cumbersome to define a proper "expected" value because of the hyperthreading feature of the machine: the black line in figure 4(c) corresponds to the number of physical core in use.



(a) *Westmere X5680@3.33 GHz* (b) *Sandybridge @2.5 GHz* (c) *Intel MIC 5110P@1.053 GHz*

Figure 4: *Multithread speedup on different processor. Points represent measurement and black line ideal scaling (in the case of MIC it represents number of physical cores involved).*

6. Applications

Even though still in development, *parmalgt* is capable of some applications.

Schrödinger functional is defined by imposing periodic boundary conditions in spatial directions and Dirichlet boundary conditions in time. In in SU(3) YangMills theory these conditions

read

$$U_k(x)\Big|_{x_0=0} = e^{C(x)} \qquad U_k(x)\Big|_{x_0=T} = e^{C'(x)}.$$

Among the different choices for C and C' one can consider constant diagonal matrices as specified in [4]; this choice induce a nontrivial background field. In perturbation theory the following decomposition holds:

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

Considering the NSPT approach we are dealing with a perturbative expansion of gluons around a nontrivial background, in this case an `AbelianBgf`. Such an expansion can be achieved by simply setting the proper 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;
```

Some preliminary results using this formulation have been presented in [5].

Another interesting application for which *parmalgt* is already providing results is the perturbative study of the Wilson flow [6], in particular the Wilson flow coupling on the lattice, defined by

$$\begin{aligned} \bar{g}^2_{GF}(L) &= \mathscr{N}^{-1}\langle t^2 E(t, T/2)\rangle|_{t=c^2L^2/8} \\ &= \mathscr{N}^{-1}\left\{\mathscr{E}^{(0)} g_0^2 + \mathscr{E}^{(1)} g_0^4 + \mathscr{E}^{(2)} g_0^6 + \ldots\right\} \end{aligned}$$

where $t > 0$ is the flow time, $E(t, T/2)$ the energy density and $\mathscr{N}$ a normalization s.t. $\bar{g}^2_{GF}(L) = g_0^2 + \mathscr{O}\left(g_0^4\right)$. A comparison with the analytical results [7] has been presented in [8] together with preliminary one and two loop computations.

7. Conclusion and outlook

In this proceeding we present a new simulation environment that aims to be flexible and general in order to allow non standard simulations on the lattice such as NSPT. This target is achieved by means of a layered structure and usage of templates.

So far some features have already been implemented: periodic and Dirichlet boundary conditions, expansions around trivial or Abelian background and Euler and 2$^{\text{nd}}$ order Runge-Kutta integrators for the Langevin and flow equations. Among different measurements it is worth to mention the Schrödinger functional and Wilson flow couplings. Multithread parallelization is properly working and multiprocess is under development. Further ongoing developments are the inclusion of fermions and improved gauge actions.

Acknowledgments

This talk was a part of a coding session sponsored partially by the PRACE-2IP project, as part of the "Community Codes Development" Work Package 8. PRACE-2IP is a 7th Framework EU

funded project (http://www.prace-ri.eu, grant agreement number: RI-283493).
This work received funding from the Research Executive Agency (REA) of the European Union under Grant Agreement number PITN-GA-2009-238353 (ITN STRONGnet) and in parts by INFN under i.s. MI11 (now QCDLAT).
M. Brambilla has been supported by MIUR (Italy) through the INFN SUMA

References

[1] G. Parisi and Y. s. Wu, "Perturbation Theory Without Gauge Fixing," Sci. Sin. **24**, 483 (1981).

[2] F. Di Renzo, G. Marchesini, P. Marenzoni and E. Onofri, "Lattice perturbation theory by Langevin dynamics," hep-lat/9308006.

[3] M. Luscher, R. Narayanan, P. Weisz and U. Wolff, "The Schrödinger functional: A Renormalizable probe for nonAbelian gauge theories," Nucl. Phys. B **384** (1992) 168 [hep-lat/9207009].

[4] M. Luscher, R. Sommer, P. Weisz and U. Wolff, "A Precise determination of the running coupling in the SU(3) Yang-Mills theory," Nucl. Phys. B **413** (1994) 481 [hep-lat/9309005].

[5] D. Hesse, M. Brambilla, M. Dalla Brida, F. Di Renzo and S. Sint, "Numerical Stochastic Perturbation Theory in the Schrödinger Functional," PoS LATTICE **2013** (2013).

[6] M. Luscher, "Trivializing maps, the Wilson flow and the HMC algorithm," Commun. Math. Phys. **293** (2010) 899 [arXiv:0907.5491 [hep-lat]].

[7] P. Fritzsch and A. Ramos, "The gradient flow coupling in the Schrödinger Functional," JHEP **1310** (2013) 008 [arXiv:1301.4388 [hep-lat]].

[8] M. Dalla Brida and D. Hesse, "Numerical Stochastic Perturbation Theory and the Gradient Flow," PoS LATTICE **2013** (2013) 326.