# PLQCD library for Lattice QCD on multi-core machines

**A. Abdel-Rehim**[*], [a] **C. Alexandrou,** [a,b] **N. Anastopoulos,**[c] **G. Koutsou,**[a] **I. Liabotis** [d] **and N. Papadopoulou**[c]

[a]*The Cyprus Institute, CaSToRC, 20 Konstantinou Kavafi Street, 2121 Aglantzia, Nicosia, Cyprus*
[b]*Department of Physics, University of Cyprus, P.O. Box 20537, 1678 Nicosia, Cyprus*
[c]*Computing Systems Laboratory, School of Electrical and Computer Engineering, National Technical University of Athens, Zografou Campus, 15773 Zografou, Athens, Greece*
[d]*Greek Research and Technology Network, 56 Mesogion Av., 11527, Athens, Greece*
*E-mail:* a.abdel-rehim@cyi.ac.cy, c.alexandrou@cyi.ac.cy, g.koutsou@cyi.ac.cy, anastop@cslab.ece.ntua.gr, iliaboti@grnet.gr, nikela@cslab.ece.ntua.gr

PLQCD is a stand–alone software library developed under PRACE for lattice QCD. It provides an implementation of the Dirac operator for Wilson type fermions and few efficient linear solvers. The library is optimized for multi-core machines using a hybrid parallelization with OpenMP+MPI. The main objectives of the library is to provide a scalable implementation of the Dirac operator for efficient computation of the quark propagator. In this contribution, a description of the PLQCD library is given together with some benchmark results.

---

[*]Speaker.

## 1. Introduction

Computer hardware for commodity clusters as well as supercomputers has evolved tremendously in the last few years. Nowadays a typical compute node has between 16 and 64 cores and possibly an accelerator such as a Graphics Processing Unit (GPU) or lately an Intel Many Integrated Core (MIC) card. This trend of packing many low-powered but massively parallel processing units is expected to continue as supercomputing technology pursues the Exascale regime. The current technology trends indicate that bandwidth to main memory will continue to lag behind computational power, which requires a rethinking of the design of lattice QCD codes such that they can efficiently run on such architectures. Taking this into account, PRACE [1] allocated resources for community code scaling activities in many computationally intensive areas including lattice QCD. The work presented here was developed under PRACE focusing on scaling codes for multi-core machines. The work we present deals with community codes, and more specifically on certain computationally intensive kernels in these codes, in order to improve their scaling and performance for multi-core architectures. We have carried out optimization work on the tmLQCD [2, 3] code and have developed a new hybrid MPI/OpenMP library (PLQCD) with optimized implementations of the Wilson Dirac kernel and a selected set of linear solvers. Our partners in this project have also performed optimization work for the Molecular Dynamics integrators used in Hybrid Monte Carlo codes, and also for Landau gauge fixing. This was done within the Chroma software suite [4] and will not be discussed here (See [5] for more information). Many other community codes of course exist but were not considered in this work (See [6] for an overview).

In what follows, we will first present the work carried out for the case of PLQCD, where we implemented the Wilson Dirac operator and associated linear algebra functions using MPI+OpenMP. In addition to using this hybrid approach for parallelism, we also implement additional optimizations such as overlapping communication and computation, using compiler intrinsics for vectorization as well as implementing the new Advanced Vector Instructions [7](AVX for Intel or QPX for Blue/Gene Q) that became recently available in new generation of processors such as the Intel Sandy-Bridge. The work done for the case of the tmLQCD package will then be presented, where we implemented some new efficient linear solvers, in particular those based on deflation such as the EigCG solver [8], for which we will give some benchmark results.

## 2. Dirac operator optimizations

A key component of the lattice Dirac operator is the hopping part given by Eq.2.1.

$$\psi(x) = \sum_{\mu=0}^{3} [U_\mu(x)(1-\gamma_\mu)\phi(x+\hat{e}_\mu) + U_\mu^\dagger(x-\hat{e}_\mu)(1+\gamma_\mu)\phi(x-\hat{e}_\mu)], \tag{2.1}$$

where, $U_\mu(x)$ is the gauge link matrix in the $\mu$ direction at site $x$, $\gamma_\mu$ are the Dirac matrices and $\hat{e}_\mu$ is a unit vector in the $\mu$ direction. $\phi$ and $\psi$ are the input and output spinors respectively. Equation 2.1 can be re-written in terms of two auxiliary fields $\theta_\mu^+(x) = (1-\gamma_\mu)\phi(x)$ and $\theta_\mu^-(x) = U_\mu^\dagger(x)(1+\gamma_\mu)\phi(x)$ as

$$\psi(x) = \sum_{\mu=0}^{3} [U_\mu(x)\theta_\mu^+(x+\hat{e}_\mu) + \theta_\mu^-(x-\hat{e}_\mu)]. \tag{2.2}$$

Because of the structure of the $\gamma$ matrices, only the upper two spin components of $\theta_\mu^\pm$ need to be computed because the lower two spin components are related to the upper ones [9]. In the following we describe some of the optimizations performed for the hopping matrix.

### 2.1 Hybrid parallelization with MPI and OpenMP

OpenMP provides a simple approach for multi-threading since it is implemented as compiler directives. One can incrementally add multi-threading to the code and also use the same code with multi-threading turned on and off. Since the main component in the hopping matrix (Dirac operator) is a large "for loop" over lattice sites, it is natural to use the for-loop parallel construct of openMP. The performance of the hybrid code is then tested against the pure MPI version. We perform a *weak scaling* test by fixing the local volume per core (or thread) and increase the number of MPI processes. The test was done on the Hopper machine at NERSC which is a Cray XE6[10]. Each compute node has 2 twelve-core AMD 'MagnyCours' at 2.1-GHz such that each 6 cores share the same cache. We find performance for the Hybrid version is maximum when assigning at most 6 threads per MPI process such that these 6 OMP threads share the same L3 cache. In Fig. 1 we show the performance of the pure MPI and the MPI+openMP with 6 threads per MPI process for a total number of cores up to 49,152 cores. From these results we first notice that using OpenMP leads to a slight degradation in performance as compared to the pure MPI case. However, as we see in the case with local volume of $12^4$, the hybrid approach performs better as we go to a large number of cores. Similar behavior has been also observed for other codes from different computational sciences (see the case studies on Hopper [11]).
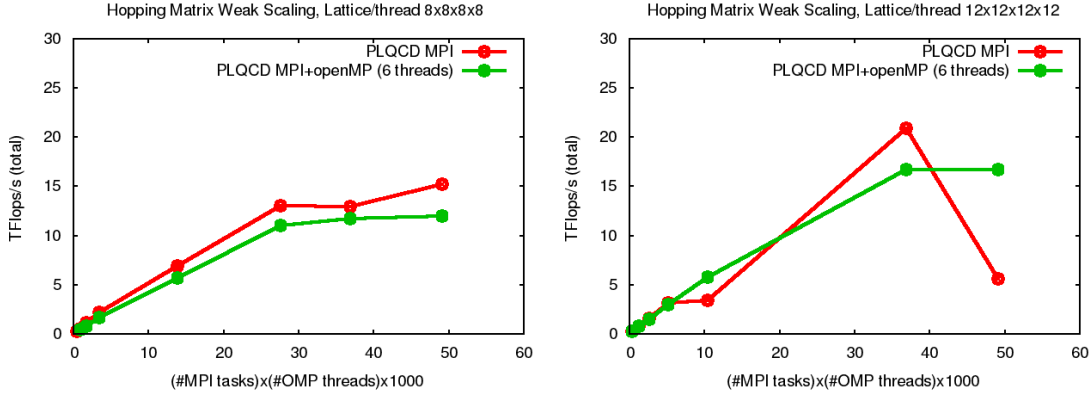


**Figure 1:** Weak scaling test for the hopping matrix on a Cray XE6 machine with local lattice volume per core $8^4$(left) and $12^4$(right).

### 2.2 Overlapping communication with computation

Typically in lattice codes one first computes the auxiliary half-spinor fields $\theta_\mu^\pm$ as given in Equation (2.2) and then communicates their values on the boundaries between neighboring processes in the $+\mu$ and $-\mu$ directions. In a blocking communication scheme, computation halts until communication of the boundaries completes. An alternative approach is to overlap communications with computations by dividing the lattice sites into bulk sites, for which nearest neighbors are

available locally, and boundary sites, for which the nearest neighbors are located on neighboring processes, and therefore can only be operated upon after communication. The order of operations for computing the result $\psi$ is then done as follows:

- Compute $\theta_\mu^+$ and begin communicating them to the neighboring MPI process in the $-\mu$ direction.

- Compute $\theta_\mu^-$ and begin communicating them to the neighboring MPI process in the $+\mu$ direction.

- Compute the result $\psi(x)$ on the bulk sites while the neighbors are being communicated.

- Wait for the communications in the $-\mu$ directions to finish, then compute the contributions $\sum_{\mu=0}^{3}[U_\mu(x)\theta_\mu^+(x+\hat{e}_\mu)]$ to the result on the boundary sites.

- Wait for the communications in the $+\mu$ directions to finish, then compute the contributions $\sum_{\mu=0}^{3}[\theta_\mu^-(x-\hat{e}_\mu)]$ on the boundary sites.

Communication is done using non-blocking MPI functions `MPI_Isend`, `MPI_Irecv` and `MPI_Wait`. A possible drawback of this approach is that one will access $\psi(x)$ and $U_\mu(x)$ in an unordered fashion different from the order it is stored in memory. This, however, can be circumvented partially by using hints in the code for prefetching. We have tested the effect of prefetching in case of sequential and random access of spinor and link fields. The test was done using a separate benchmark kernel code which isolates the link-spinor multiplication. As can be seen in Fig. 3, prefetching becomes important for a large number of sites, i.e. when data (spinors and links) can not fit in the cache memory, which is a typical situation for lattice calculations. It is also noted that accessing the sites randomly reduces the performance, as would be expected. In this case one can improve the situation by defining a pointer array, e.g. for the spinors $\psi'(\mathtt{i}) = \&\psi(\mathtt{x}[\mathtt{i}])$ where $\mathtt{x}[\mathtt{i}]$ is the site to be accessed at step $\mathtt{i}$ in the loop such as we show in pseudo-code in Fig. 2. These pointers can be defined *a priori*. This improves the predictive ability of the hardware as is shown in Fig. 3 where we compare the different prefetching and addressing schemes.

```
Sequential access          Random access              Random access with pointers
 for(i=0; i< N; i++)  for(i=0; i< N; i++)   /* prepare pointer arrays */
    ψ(i) = U(i)φ(i)         ψ(x[i]) = U(x[i])φ(x[i])  for(i=0; i< N; i++)
                                                 ψ'(i) = &ψ(x[i])
                                                 U'(i) = &U(x[i])
                                                 φ'(i) = &φ(x[i])
                                            for(i=0; i< N; i++)
                                                 *ψ'(i) = *U'(i) * φ'(i)
```

**Figure 2:** Pseudo-code describing the site access for the case of simple sequential access (left), random access using an index array $\mathtt{x}[\mathtt{i}]$ (centre) and random access using precalculated pointer arrays (right).
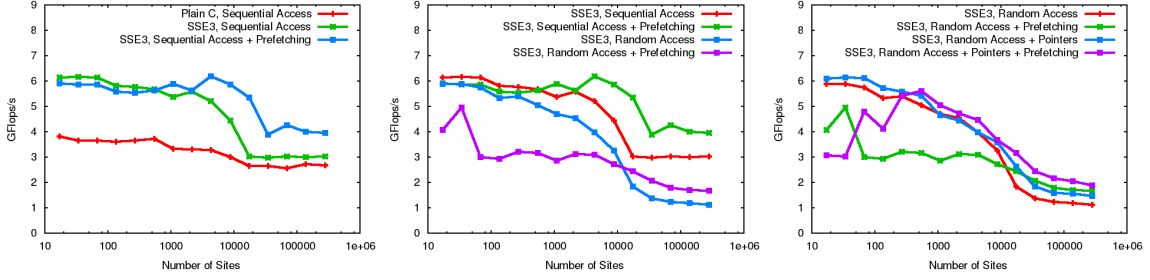
**Figure 3:** Effect of prefetching for the cases of sequential (left) and random access (centre) of lattice sites. In the right panel we compare the effect of having precalculated pointer arrays for the case of random access.

## 2.3 Vectorization using AVX instructions

Lattice QCD codes benefit from the vectorization capabilities of modern processors starting from SSE to SSE3. In SSE3, we have 16 vector registers known as XMM registers, each 128-bit wide thus able to store either 2 double precision floating point variables or 4 single precision. The AVX extensions are extensions to the x86 instruction set implemented both by Intel and AMD. These first appeared in 2011 in the Intel SandyBridge processors and later by AMD in their Bulldozer processor. The 16 XMM registers of SSE3 are now 256-bit wide and known as YMM registers. AVX-capable floating point units are able to perform on 4 double precision floating point numbers or 8 single precision. Implementing these extensions in the vectorized parts of lattice codes has the potential of providing a gain of up to a factor 2 in an ideal situation, although in practice this depends on the layout of lattice data. We provided an implementation of these extensions using inline intrinsics. In this implementation a single SU(3) matrix multiplies two SU(3) vectors simultaneously. A gain of about a factor of 1.5 is achieved for the hopping matrix in the tmLQCD code in double precision as shown in Fig.(4). For illustration, a code snippet for multiplying two complex numbers by two complex numbers using AVX is shown in Fig.(5).

## 3. EigCG solver for Twisted-Mass fermions

Twisted-Mass fermions offer the advantage of automatic $O(a)$ improvement when tuned to maximal twist [12]. Within this development work we have added an incremental deflation algorithm, known as EigCG, to the tmLQCD package. Numerical tests showed a considerable speed-up of the solution of the linear systems on the largest volumes simulated by the European Twisted Mass Collaboration (ETMC). For illustration, we show in Fig.(6) the time to solution with EigCG on a Twisted-Mass configuration with 2+1+1 dynamical flavors with lattice size $48^3 \times 96$ at $\beta = 2.1$, and pion mass $\approx 230$ MeV. In this case the total number of eigenvectors deflated was 300 which was built incrementally by computing 10 eigenvectors during the solution of the first 30 right-hand sides using a search subspace of size 60. All systems are solved in double precision to relative tolerance of $10^{-8}$.
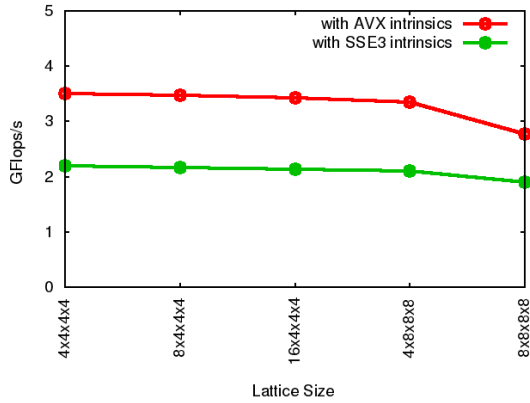
**Figure 4:** Comparing the performance of the hopping matrix of tmLQCD using SSE3 and AVX in double precision on an Intel SandyBridge processor.

```
#include <immintrin.h>
/* t0:  a+b*I, e+f*I and t1:  c+d*I, g+h*I
 * return:  (ac-bd) + (ad+bc)*I,
 * (eg-fh) + (eh+fg)*I */
static inline __m256d
complex_mul_regs_256(__m256d t0, __m256d t1)
{
 __m256d t2;
 t2 = t1;
 t1 = _mm256_unpacklo_pd(t1,t1);
 t2 = _mm256_unpackhi_pd(t2,t2);
 t1 = _mm256_mul_pd(t1, t0);
 t2 = _mm256_mul_pd(t2, t0);
 t2 = _mm256_shuffle_pd(t2, t2, 5);
 t1 = _mm256_addsub_pd(t1, t2);
 return t1;
}
```

**Figure 5:** Multiplying two complex numbers by two complex number of type double using AVX instructions.
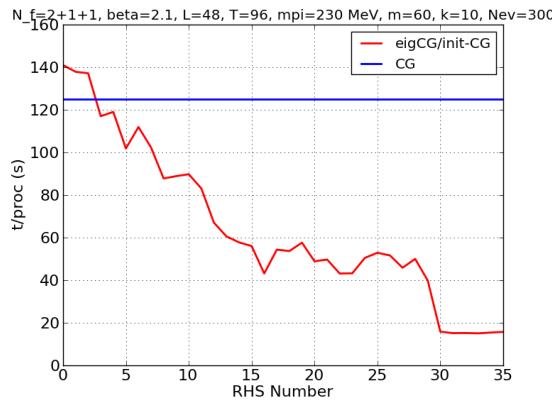


**Figure 6:** Solution time per process for the first 35 right-hand sides using Incremental EigCG as compared to CG on a Twisted-Mass configuration with lattice size $48^3 \times 96$ at $\beta = 2.1$, and pion mass $\approx 230$ MeV.

## 4. Conclusions and Summary

We have carried out development effort for a few selected kernels used in lattice QCD. The first of these efforts included the development of a hybrid MPI/OpenMP library which includes parallelized kernels for the Wilson Dirac operator and few associated solvers. A number of parallelization strategies have been investigated, such as for overlapping communication with computations. The code has been shown to scale fairly well on the Cray XE6. In terms of single process performance, we carried out initial vectorization efforts for AVX where we see an improvement of $\sim 1.5$ compared to the ideal 2. In addition we have investigated several data-ordering and associated prefetching strategies.

For the case of tmLQCD, the main software code of the ETMC collaboration, we have imple-

mented an efficient linear solver which incrementally deflated the twisted-mass Dirac operator to give a speed-up of about 3 times when enough right-hand-sides are required. This is already in use in production projects, such as in Refs. [14] and [13].

All codes are publicly available. PLQCD is available through the HPCFORGE website at the Swiss National Supercomputing Centre (CSCS) where more information is available within the code documentation. Our EigCG implementation in tmLQCD is available via git-hub.

**Acknowledgements**

# References

[1] http://www.prace-ri.eu/.

[2] K. Jansen and C. Urbach, *Comput.Phys.Commun.* **180**, 2717 (2009), [arXiv:0905.3331].

[3] ETM Collaboration, https://github.com/etmc/tmLQCD.

[4] http://usqcd.jlab.org/usqcd-docs/chroma/.

[5] See the public deliverable D8.3 on the PRACE website under PRACE-2IP.

[6] A. Deuzeman, PoS(LATTICE 2013).

[7] See the Intel Developer manual.

[8] A. Stathopoulos and K. Orginos, *Computing and deflating eigenvalues while solving multiple right-hand side linear systems with an application to quantum chromodynamics*, *SIAM J. Sci. Comput.* 2010; **32(1)**:439–462, [arXiv:0707.0131].

[9] See for example the documentation of the $DD - HMC$ code by M. Lüscher.

[10] The Hopper Cray XE6 machine at NERSC.

[11] See documentation for combining MPI and openMP on the NERSC website.

[12] R. Frezzotti *et al.* [Alpha Collaboration], *Lattice QCD with a chirally twisted mass term*, *JHEP* **0108**, 058 (2001) [hep-lat/0101001].

[13] C. Alexandrou, M. Constantinou, S. Dinter, V. Drach, K. Hadjiyiannakou, K. Jansen, G. Koutsou and A. Vaquero, arXiv:1309.7768 [hep-lat].

[14] A. Abdel-Rehim, C. Alexandrou, M. Constantinou, V. Drach, K. Hadjiyiannakou, K. Jansen, G. Koutsou and A. Vaquero, arXiv:1310.6339 [hep-lat].

PoS(LATTICE 2013)419