PROCEEDINGS OF SCIENCE

Three Dirac operators on two architectures with one piece of code and no hassle

Stephan Dürr*
University of Wuppertal, D-42119 Wuppertal, Germany
IAS/JSC Forschungszentrum Jülich, D-52425 Jülich, Germany
E-mail: durr (AT) itp.unibe.ch

A simple minded approach to implement three discretizations of the Dirac operator (staggered, Wilson, Brillouin) on two architectures (KNL and core i7) is presented. The idea is to use a high-level compiler along with OpenMP parallelization and SIMD pragmas, but to stay away from cache-line optimization and/or assembly-tuning. The implementation is for N_v right-hand-sides, and this extra index is used to fill the SIMD pipeline. On one KNL node single precision performance figures for $N_c = 3$, $N_v = 12$ read 475 Gflop/s, 345 Gflop/s, and 790 Gflop/s for the three discretization schemes, respectively.

The 36th Annual International Symposium on Lattice Field Theory - LATTICE2018
22-28 July, 2018
Michigan State University, East Lansing, Michigan, USA.

*Speaker.

1. Introduction

Recent years brought plenty of machines with peak performances in the multi-petaflop/s range, but it gets increasingly difficult to achieve good strong-scaling behavior with actual scientific codes. Lattice QCD is still in a fortunate position to harness these capabilities [1, 2, 3], as it does not require any run-time dependent data structures. On the other hand, an un-optimized non-parallel code tends to have $O(10^5)$ lines. This means that significant human resources must be spent to parallelize a lattice code and to obtain good performance figures on a given architecture.

Ideally one would have a piece of code, written in a high-level language, which parallelizes and reaches decent (read: non-optimal but non-disastrous) performance upon compilation on a given new architecture. In these proceedings I report on an attempt to do this on the single-node level, based on OpenMP (OMP) threads and again OpenMP pragmas for SIMD-pipelining. I concentrate on the part which takes most time in actual computations – the matrix-times-vector operation for a given Dirac operator, considering the Susskind ("staggered"), Wilson and Brillouin varieties.

2. Vector layout options

The routines are written in Fortran 2008, using the stride notation, as this allows for compact source files (like in matlab). The gauge field U is defined as a 7-dimensional array through `complex(kind=sp),dimension(Nc,Nc,4,Nx,Ny,Nz,Nt) :: U`, with parameters like `Nc=3` and `sp,dp` (for single and double precision, respectively) specified at compile time. Hence `U(:,:,3,x,y,z,t)` defines N_c^2 complex numbers, arranged contiguously in memory.

With Wilson-type vectors arranged in blocks of N_v right-hand sides, the layout options include `vec(Nc,4,Nv,...)`, `vec(4,Nc,Nv,...)`, `vec(Nc,Nv,4,..)`, `vec(Nv,Nc,4,..)`, `vec(4,Nv,Nc,...)`, `vec(Nv,4,Nc,...)`, where the dots stand for `Nx*Ny*Nz*Nt`. This restriction of having the space-time index as the slowest (right-most) index precludes sophisticated SIMD strategies (see Refs. [1, 2]), but it may facilitate the use of PGAS concepts (see below). With Susskind-type vectors arranged in blocks of N_v right-hand sides, the layout options under the same restriction are `suv(Nc,Nv,Nx*Ny*Nz*Nt)`, `suv(Nv,Nc,Nx*Ny*Nz*Nt)`.

Our task is to optimize the performance under the self-imposed set of restrictions. An important ingredient in the code is that all contributions to the "out" vector are collected in the thread-private variable `site`, which for each space-time index is written *once*. This avoids write collisions among threads in a natural way. We have the same 6 layout options as for `vec` to define `site` as an array of dimension 3 in the Wilson case (or the same 2 options in the staggered case).

3. Staggered kernel details and performance

The Susskind ("staggered") Dirac operator is defined through

$$D_{\rm S}(x,y) = \sum_\mu \eta_\mu(x)\,\frac{1}{2}\,[V_\mu(x)\delta_{x+\hat\mu,y} - V_\mu^\dagger(x-\hat\mu)\delta_{x-\hat\mu,y}] \tag{3.1}$$

with $\eta_1(x)=1$, $\eta_2(x)=(-1)^{x_1}$, $\eta_3(x)=(-1)^{x_1+x_2}$, $\eta_4(x)=(-1)^{x_1+x_2+x_3}$. Here $V_\mu(x)$ represents a smeared version of the (original) gauge link $U_\mu(x)$, i.e. a gauge-covariant parallel transporter from

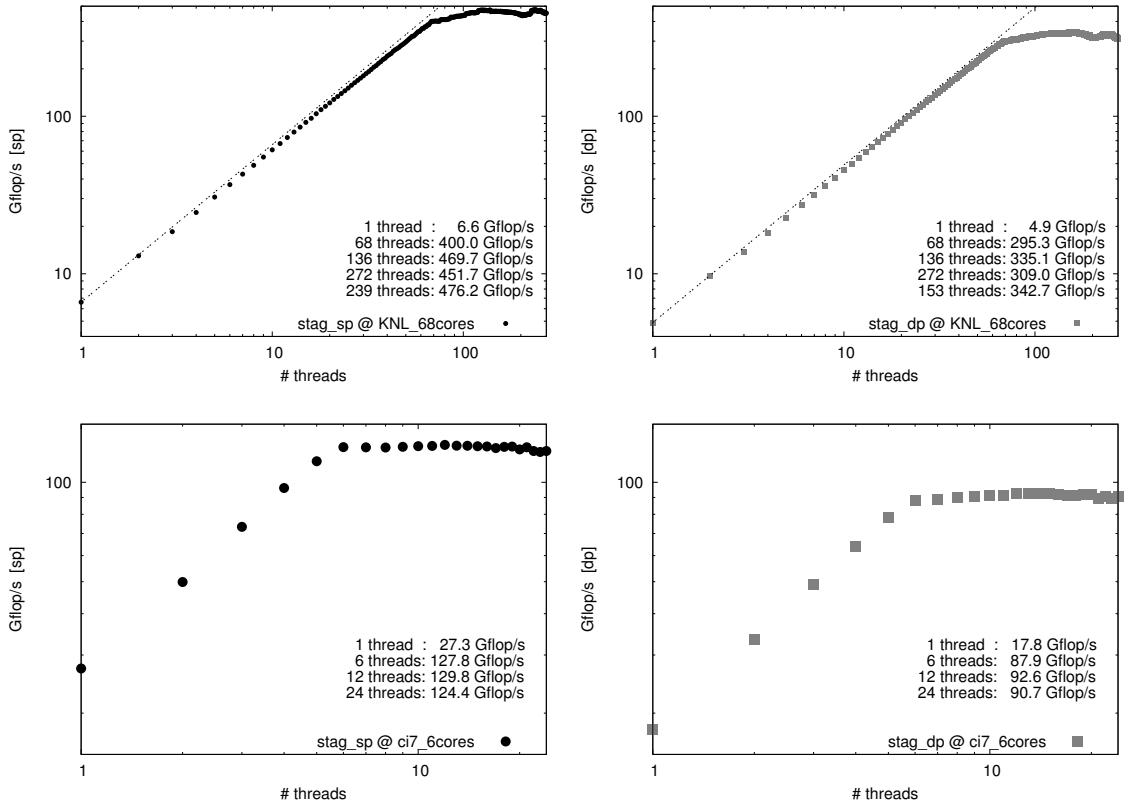

Figure 1: Log-log scaling plot of the single processor performance in Gflop/s versus the number of OMP threads for the staggered Dirac operator in `sp` (left) and `dp` (right). The top panels feature a KNL processor with 68 cores, the bottom panels a Broadwell chip with 6 cores. The lattice size is $34^3 \times 68$.

$x+\hat{\mu}$ to x, to reduce taste-symmetry breaking. From a HPC viewpoint, a clear advantage of this operator with precomputed V_μ is that its stencil is restricted to sites which are at most one hop away. Still, it is not trivial to reach an acceptable performance on a many-core architecture [4, 5].

In our framework we have 2 options for the `suv` layout, 2 options for `site`, and 2 reasonable loop nestings. It is thus possible to implement all these options, and to compare the timings. For one choice (the one performing best on the KNL architecture) thread scaling results are shown in Fig. 1. Performance on the Broadwell architecture seems far less sensitive to these choices.

4. Wilson kernel details and performance

The Wilson Dirac operator is defined through

$$D_{\rm W}(x,y) = \sum_\mu \gamma_\mu \nabla^{\rm std}_\mu(x,y) - \frac{a}{2}\triangle^{\rm std}(x,y) + m_0\delta_{x,y} - \frac{c_{\rm SW}}{2}\sum_{\mu<\nu}\sigma_{\mu\nu}F_{\mu\nu}\delta_{x,y}\,, \tag{4.1}$$

where $\nabla^{\rm std}_\mu$ is a 2-point discretization of the covariant derivative

$$a\nabla^{\rm std}_\mu(x,y) = \frac{1}{2}\left[V_\mu(x)\delta_{x+\hat{\mu},y} - V_{-\mu}(x)\delta_{x-\hat{\mu},y}\right] \tag{4.2}$$

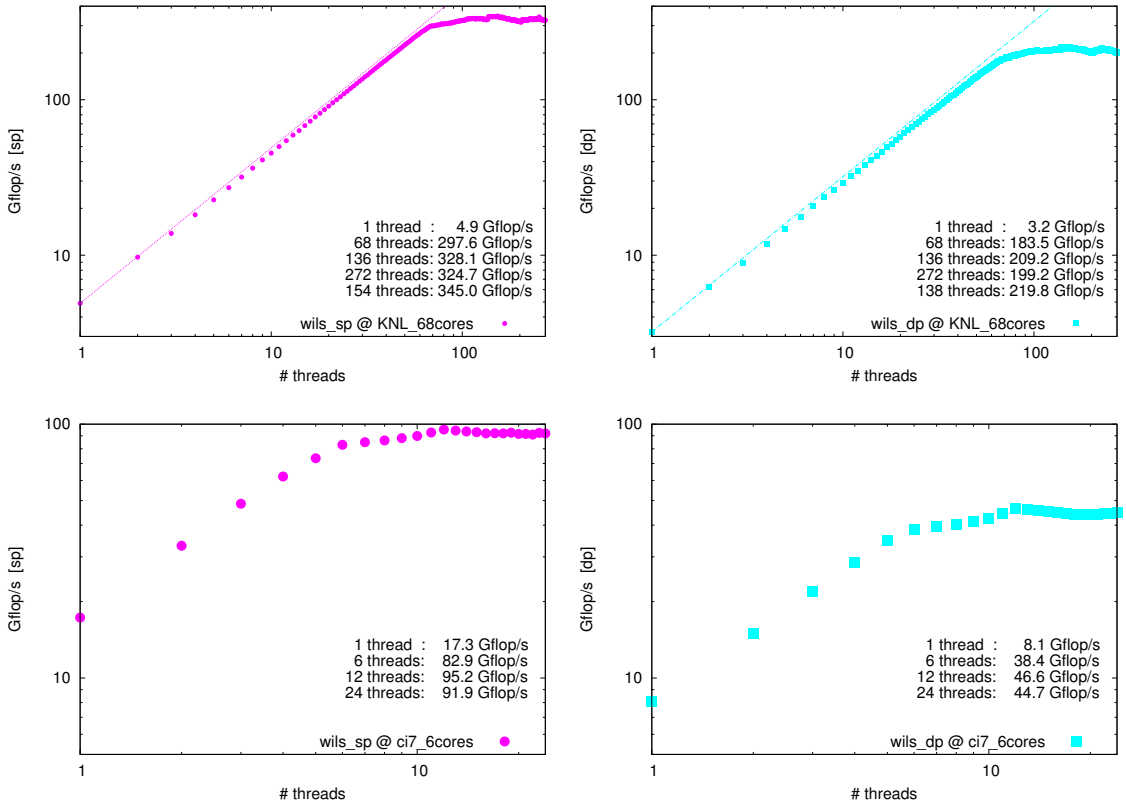

Figure 2: Log-log scaling plot of the single processor performance in Gflop/s versus the number of OMP threads for the Wilson Dirac operator at $c_{\rm SW} = 0$ in `sp` (left) and `dp` (right). The top panels feature a KNL processor with 68 cores, the bottom panels a Broadwell chip with 6 cores. The lattice size is $34^3 \times 68$.

and $\triangle^{\rm std}$ is a 9-point discretization of the covariant Laplacian (sum over 4 pos. and 4 neg. indices)

$$a^2 \triangle^{\rm std}(x,y) = -8\,\delta_{x,y} + 1 \sum_\mu V_\mu(x)\delta_{x+\hat{\mu},y}\,. \tag{4.3}$$

Also this operator is HPC friendly, since its stencil contains at most 1-hop terms.

In our framework we have 6 options for the `vec` layout, times 6 options for `site`, times a few reasonable loop nestings. For the standard Laplacian $\triangle^{\rm std}$ in the Wilson operator the latter set of options is 4, resulting in a total of 144 routines. Evidently, this brings some limitations to the "best of breed" ansatz, since it may be a little awkward to test each routine with each possible thread count (e.g. 1 through 272 on the KNL). In practice, it seems reasonable to restrict the selection process to just a few thread counts (e.g. 68, 136, and 272 on the KNL).

Full thread scaling results for one such choice are displayed in Fig. 2. Similar to Fig. 1, we see an almost linear increase in performance up to 68 threads on the KNL. This is followed by another linear gain (albeit with a smaller slope) up to 136 threads. Beyond that point results wiggle a bit out to 272 threads. The maximum appears in a number of threads (154 in `sp`, 138 in `dp`) which seems hard to predict. Again, the code seems to perform (without any change) reasonably well on the Broadwell architecture, too. Having more than 2 threads per core does not enhance performance, but the good news is that it's not really detrimental either. Next steps of performance tuning would naturally include eo-decomposition and gauge compression (from N_c to $N_c - 1$ columns).

#hop	#terms	#paths	formula
1	8	1!=1	$W_\mu(x) = V_\mu(x)$ (smeared link, $\mu \in \{\pm1,\pm2,\pm3,\pm4\}$)
2	24	2!=2	$W_{\mu+\nu}(x) = \frac{1}{2}[V_\mu(x)V_\nu(x+\hat{\mu}) + \text{perm}]$
3	32	3!=6	$W_{\mu+\nu+\rho}(x) = \frac{1}{6}[V_\mu(x)V_\nu(x+\hat{\mu})V_\rho(x+\hat{\mu}+\hat{\nu}) + \text{perms}]$
4	16	4!=24	$W_{\mu+\nu+\rho+\sigma}(x) = \frac{1}{24}[V_\mu(x)V_\nu(x+\hat{\mu})V_\rho(x+\hat{\mu}+\hat{\nu})V_\sigma(x+\hat{\mu}+\hat{\nu}+\hat{\sigma}) + \text{perms}]$

Table 1: Overview of the set of off-axis links $W_{\text{dir}}(x)$, with lengths ranging from 1 to 4 hops. Given a site x, 81 directions are possible, but one is trivial, and the remaining 80 can be reduced to 40 based on $W_{-\text{dir}}(x) = W^\dagger_{\text{dir}}(x-\text{dir})$. In the code W is precomputed and stored in the array `W(Nc,Nc,40,Nx,Ny,Nz,Nt)`. Note that for 36 of the 40 directions the entry is not special unitary, and no gauge compression is possible.

5. Brillouin kernel details and performance

The Brillouin Dirac operator is defined through [6]

$$D_{\text{B}}(x,y) = \sum_\mu \gamma_\mu \nabla^{\text{iso}}_\mu(x,y) - \frac{a}{2}\triangle^{\text{bri}}(x,y) + m_0\delta_{x,y} - \frac{c_{\text{SW}}}{2}\sum_{\mu<\nu}\sigma_{\mu\nu}F_{\mu\nu}\delta_{x,y}\,, \tag{5.1}$$

where the isotropic derivative ∇^{iso}_μ is a 54-point discretization of the covariant derivative

$$\begin{aligned} a\nabla^{\text{iso}}_\mu(x,y) \;=\; & \rho_1\,[W_\mu(x)\delta_{x+\hat{\mu},y} - W_{-\mu}(x)\delta_{x-\hat{\mu},y}] \\ & +\; \rho_2 \sum\nolimits_{\neq(\nu;\mu)}[W_{\mu+\nu}(x)\delta_{x+\hat{\mu}+\hat{\nu},y} - (\mu\to-\mu)] \\ & +\; \rho_3 \sum\nolimits_{\neq(\nu,\rho;\mu)}[W_{\mu+\nu+\rho}(x)\delta_{x+\hat{\mu}+\hat{\nu}+\hat{\rho},y} - (\mu\to-\mu)] \\ & +\; \rho_4 \sum\nolimits_{\neq(\nu,\rho,\sigma;\mu)}[W_{\mu+\nu+\rho}(x)\delta_{x+\hat{\mu}+\hat{\nu}+\hat{\rho}+\hat{\sigma},y} - (\mu\to-\mu)] \end{aligned} \tag{5.2}$$

and the Brillouin Laplacian $\triangle^{\text{bri}}$ is a 81-point discretization of the covariant Laplacian

$$\begin{aligned} a^2\triangle^{\text{bri}}(x,y) = \lambda_0\,\delta_{x,y} \;+\; & \lambda_1 \sum\nolimits_\mu W_\mu(x)\delta_{x+\hat{\mu},y} \\ & +\; \lambda_2 \sum\nolimits_{\neq(\mu,\nu)} W_{\mu+\nu}(x)\delta_{x+\hat{\mu}+\hat{\nu},y} \\ & +\; \lambda_3 \sum\nolimits_{\neq(\mu,\nu,\rho)} W_{\mu+\nu+\rho}(x)\delta_{x+\hat{\mu}+\hat{\nu}+\hat{\rho},y} \\ & +\; \lambda_4 \sum\nolimits_{\neq(\mu,\nu,\rho,\sigma)} W_{\mu+\nu+\rho+\sigma}(x)\delta_{x+\hat{\mu}+\hat{\nu}+\hat{\rho}+\hat{\sigma},y} \end{aligned} \tag{5.3}$$

with $(\rho_1,\rho_2,\rho_3,\rho_4) \equiv (64,16,4,1)/432$ and $(\lambda_0,\lambda_1,\lambda_2,\lambda_3,\lambda_4) \equiv (-240,8,4,2,1)/64$ respectively. In (5.2) the last sum extends over (pos. and neg.) indices (ν,ρ,σ) which are mutually unequal and different from μ. In (5.3) the last sum extends over indices (μ,ν,ρ,σ) which are pairwise unequal. In these formulas $W_{\text{dir}}(x)$ denotes a link in direction "dir" which may be on-axis (dir=μ) or off-axis with length $\sqrt{2}$ (dir=$\mu\nu$) or $\sqrt{3}$ (dir=$\mu\nu\rho$) or $\sqrt{4}$ (dir=$\mu\nu\rho\sigma$). More details are given in Tab. 1.

The Brillouin operator (5.1) brings new perspectives on PDFs [6, 7] and – when used in conjunction with the overlap procedure [8, 9] – on heavy-quark physics [10, 11]. From a HPC viewpoint the Brillouin operator is interesting, since its computational intensity is by a factor 2.5 higher than for the Wilson operator. The choice of treating N_v right-hand-sides simultaneously enhances the computational intensity of either operator, while keeping this ratio almost invariant [11]. In a very distant future, when cycles are totally irrelevant, one might opt for *not* precomputing W; this would trigger a massive enhancement of the computational intensity of the operator (5.1).

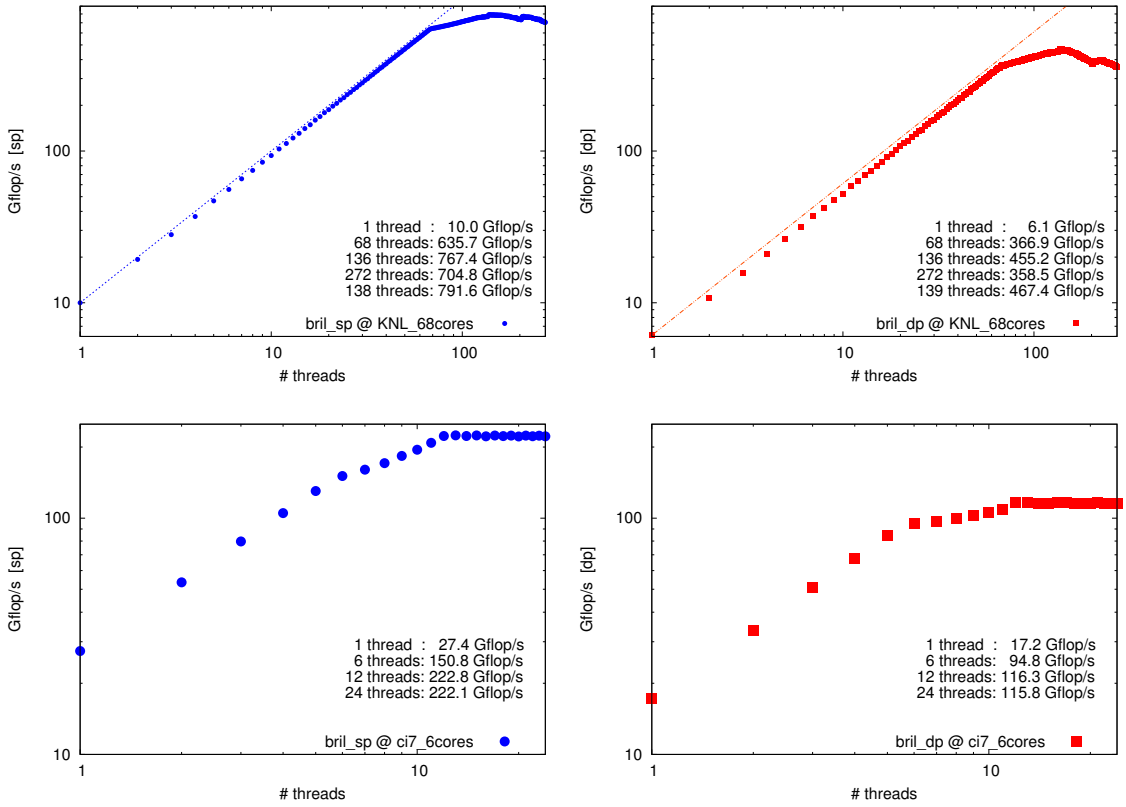

Figure 3: Log-log scaling plot of the single processor performance in Gflop/s versus the number of OMP threads for the Brillouin Dirac operator at $c_{\rm SW} = 0$ in `sp` (left) and `dp` (right). The top panels feature a KNL processor with 68 cores, the bottom panels a Broadwell chip with 6 cores. The lattice size is $34^3 \times 68$.

Thread scaling results are shown in Fig. 3. As in previous cases, we see nearly-perfect scaling behavior up to 68 threads on the KNL. After this, there is another performance increase up to 136 threads. Beyond that point, performance figures wiggle a bit out to 272 threads. And again, the (unchanged, just recompiled) code performs reasonably well on the Broadwell architecture, too. Unlike in the Wilson case, the author is unaware of simple recipes for further improvement.

6. Summary and outlook

In summary, thread scaling results for the Susskind, Wilson and Brillouin varieties of the Dirac operator in lattice QCD were presented. They are based on straightforward implementations in Fortran 2008, with OpenMP pragmas for shared-memory parallelization and SIMD pipelining.

No cache-access optimization and no hand-assembly tuning have been applied, and still reasonable performance figures can be obtained. Key to the improvement over last year's version [12] is an ansatz where performance critical routines are written for a variety of layouts of the in/out-vectors, of accumulation variables, and possibly loop nestings. For a given architecture and compiler combination, all of these routines are compiled "out of the box", and a few test calls will quickly reveal which option features best on a particular machine. With this "best of breed" ansatz, the winning combination is subsequently used for actual computations.

The plots presented in this contribution illustrate that this strategy proves successful on the single-node level. What would be important for actual calculations, however, is a working concept (along these lines) on the multi-node level. The author's hope is that vendors will finally provide efficient PGAS (Coarray Fortran and/or Unified Parallel C) support for CPUs. In fact, this is the rationale for not sacrificing any of the space-time indices for the SIMD vectorization. In the event the GPU world offers this feature more promptly, this will be a clear case for switching to OpenACC.

Acknowledgements: Program development and test runs were performed on the DEEP-ER system at IAS/JSC in Jülich. The author likes to thank Eric Gregory for useful discussion.

References

[1] P. A. Boyle, PoS LATTICE **2016**, 013 (2017) doi:10.22323/1.256.0013 [arXiv:1702.00208 [hep-lat]].

[2] A. Rago, EPJ Web Conf. **175**, 01021 (2018) doi:10.1051/epjconf/201817501021 [arXiv:1711.01182].

[3] M. Lin, "Machines and Algorithms for Lattice QCD," talk at Lattice 2018 (these proceedings).

[4] C. DeTar, D. Doerfler, S. Gottlieb, A. Jha, D. Kalamkar, R. Li and D. Toussaint, PoS LATTICE **2016**, 270 (2016) doi:10.22323/1.256.0270 [arXiv:1611.00728 [hep-lat]].

[5] C. DeTar, S. Gottlieb, R. Li and D. Toussaint, EPJ Web Conf. **175**, 02009 (2018). doi:10.1051/epjconf/201817502009 [arXiv:1712.00143 [hep-lat]].

[6] S. Durr and G. Koutsou, Phys. Rev. D **83**, 114512 (2011) [arXiv:1012.3615 [hep-lat]].

[7] S. Durr, G. Koutsou and T. Lippert, Phys. Rev. D **86**, 114514 (2012) [arXiv:1208.6270 [hep-lat]].

[8] H. Neuberger, Phys. Lett. B **417**, 141 (1998) [hep-lat/9707022].

[9] H. Neuberger, Phys. Lett. B **427**, 353 (1998) [hep-lat/9801031].

[10] Y. G. Cho, S. Hashimoto, A. Juttner, T. Kaneko, M. Marinkovic, J. I. Noaki and J. T. Tsang, JHEP **1505**, 072 (2015) [arXiv:1504.01630 [hep-lat]].

[11] S. Durr and G. Koutsou, arXiv:1701.00726 [hep-lat].

[12] S. Durr, EPJ Web Conf. **175**, 02001 (2018) doi:10.1051/epjconf/201817502001 [arXiv:1709.01828].