$O(a)$-improved QCD+QED Wilson Dirac operator on GPUs

openQxD with QUDA

Roman Gruber

RC* collaboration: Anian Altherr Isabel Campos-Plasencia, Alessandro Cotellucci, Alessandro De Santis, Roman Gruber, Tim Harris, Javad Komijani, Marina Marinkovic, Francesca Margari, Letizia Parato, Agostino Patella, Gaurav Ray, Sara Rosso, Nazario Tantalo, and Paola Tavella

July 30, 2024

Intro / Overview

Motivation

OPENQXD [5]

- Simulations of QCD and QCD+QED $O(a)$ improved Wilson-Clover fermions
- Based on openQCD v1.6 [1, 2]
- Variety of BCs; open/SF/periodic in time, $C^\star$ boundaries [3] or periodic boundaries in space
- Powerful solvers: CGNE, GCR with Schwarz-alternating procedure and inexact deflation [4]
- Pure-MPI parallelisation, C89 standard (next release will be C99)
- Actively developed and maintained by $RC^\star$ collaboration

Requirement

$C^\star$ boundaries and QCD+QED Wilson-Clover fermions

Main Goal

Offload solves to GPU (target system: new Alps machine and Lumi-G)

Main Goal

Offload solves to GPU (target system: new Alps machine and Lumi-G)

→ OpenMP offloading

- + Easy and rapid porting
- − Disappointing results (more efforts required)

Main Goal

Offload solves to GPU (target system: new Alps machine and Lumi-G)

- → OpenMP offloading
  - + Easy and rapid porting
  - − Disappointing results (more efforts required)
- → Coupling to existing solver suite
  - − Operator on GPU still a problem
  - − General solvers not on eye level with state-of-the-art lattice solvers

Main Goal

Offload solves to GPU (target system: new Alps machine and Lumi-G)

- → OpenMP offloading
  - + Easy and rapid porting
  - − Disappointing results (more efforts required)
- → Coupling to existing solver suite
  - − Operator on GPU still a problem
  - − General solvers not on eye level with state-of-the-art lattice solvers
- → Own CUDA/HIP implementation in openQxD
  - + Cleanest solution (no external dependencies)
  - − Insane effort (lots of core changes, breaking changes, ...)

GPU-enabled openQxD - Motivation

Main Goal

Offload solves to GPU (target system: new Alps machine and Lumi-G)

→ OpenMP offloading
- + Easy and rapid porting
- − Disappointing results (more efforts required)

→ Coupling to existing solver suite
- − Operator on GPU still a problem
- − General solvers not on eye level with state-of-the-art lattice solvers

→ Own CUDA/HIP implementation in openQxD
- + Cleanest solution (no external dependencies)
- − Insane effort (lots of core changes, breaking changes, ...)

→ Coupling to QUDA
- + No need to reinvent the wheel
- + Get all features of QUDA (solver suite, eigensolvers, ...)
- − Only real additional efforts: (1) Interface, (2) $C^\star$ boundaries, (3) QCD+QED Wilson-Clover

QUDA [8]

- Plug and play library to offload Dirac solves
- Supports many lattice discretisations (Wilson, staggered, Domain-wall, ...)
- Powerful solvers: BiCGstab, GCR with multigrid [6, 7], ...
- C++-14 standard
- Supports NVIDIA, AMD, Intel and CPU threading
- Actively developed and maintained by NVIDIA + many others
- NVIDIA licence (similar to MIT)

Interfacing openQxD with QUDA

```c
/* Complex double struct */
typedef struct
{
   double re,im;
} complex_dble;
```

Figure: Complex double struct

```c
/* Clover field struct */
typedef struct
{
   double u[36];
} pauli_dble;
```

Figure: Clover field struct

```c
/* Gauge field struct */
typedef struct
{
   complex_dble c11,c12,c13,c21,c22,c23,c31,c32,c33;
} su3_dble;
```

Figure: Gauge field struct

- Gauge field d.o.f: $4V$ (V = lattice volume, 8 directions)
- Clover field d.o.f: $2V$ (V, 2 chiralities, 6x6 matrix (complex, Hermitian))

```c
typedef struct
{
    complex_dble c1,c2,c3;
} su3_vector_dble;
```

Figure: SU(3) vector struct

```c
typedef struct
{
    su3_vector_dble c1,c2,c3,c4;
} spinor_dble;
```

Figure: Spinor field struct

- Spinor field d.o.f: V (V = lattice volume, 4 spin, 3 color) $\longrightarrow$ array of structs

Different gauge field layouts

- openQxD
  - stores 8 (forward and backward) directed gauge fields for all odd-parity points
  - locally stores gauge fields on the boundaries only for odd-parity points and not for even-parity points
- QUDA
  - 4 gauge fields for each space-time point (one for each positive direction

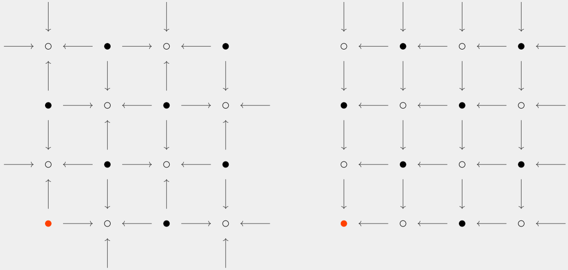

Figure: 2D example (4×4 local lattice) of how and which gauge fields are stored in memory in openQxD (left) and QUDA (right). Filled lattice points are even, unfilled odd lattice points.

Status

Interface

$C^\star$ boundaries

QCD+QED Wilson-Clover

Status

- ✅ Interface
- $C^\star$ boundaries
- QCD+QED Wilson-Clover

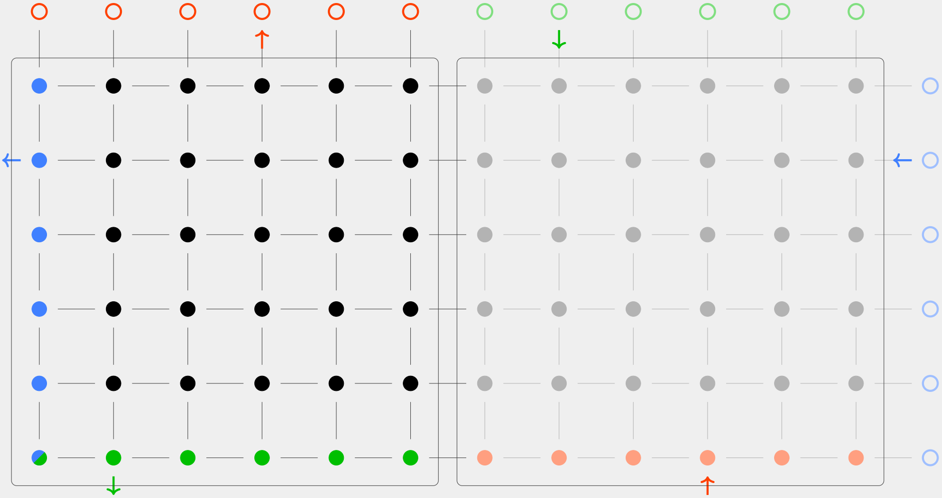

Figure: 2D example of a 6×6 lattice with $C^\star$ boundary conditions on both directions. We have the (doubled) x-direction (horizontal) and a direction with $C^\star$ boundaries (vertical). Left is the physical, right the mirror lattice. The union is the extended lattice

$C^\star$ boundaries: Implementation in QUDA

- Analogue to the implementation in openQCD
- Doubling the lattice as it comes from openQxD (i.e. additional index: physical, mirror)
- Communication grid topology struct now contains a member property `cstar` $\longrightarrow$ number of spatial $C^\star$ directions
- `comm_rank_displaced( )`: calculates the neighbouring rank number given one of (positive or negative) 8 directions $\longrightarrow$ implements the shifted boundaries

Status

- ✔ Interface
- $C^\star$ boundaries
- QCD+QED Wilson-Clover

STATUS

- ✅ Interface
- ✅ $C^\star$ boundaries
- QCD+QED Wilson-Clover

QCD+QED

- In addition to the $SU(3)$-valued gauge field $U_\mu(x)$, we have the $U(1)$-valued gauge field $A_\mu(x)$
- Combined: $U(3)$-valued field $e^{iqA_\mu(x)}U_\mu(x)$ with q_f the charge of a quark
- In QUDA, we just use
  - `QUDA_RECONSTRUCT_9`
  - `QUDA_RECONSTRUCT_13`
  - `QUDA_RECONSTRUCT_NO`
- We have an $U(1)$ SW-term,

$$D_{\mathrm{w}} \to D_{\mathrm{w}} + qc_{\mathrm{sw}}^{U(1)} \frac{i}{4} \sum_{\mu,\nu=0}^{3} \sigma_{\mu\nu} \hat{A}_{\mu\nu} \,, \tag{1}$$

where q is the charge and the $U(1)$ and $\hat{A}_{\mu\nu}(x)$ is the field strength tensor.

QCD+QED: Implementation in QUDA

- Resulting term has the same properties as the $SU(3)$ SW-term (Hermitian, diagonal w.r.t chiralities)
- Clover field reorder class:
  openQxD (row-major):

$$\begin{pmatrix} u_0 & u_6+iu_7 & u_8+iu_9 & u_{10}+iu_{11} & u_{12}+iu_{13} & u_{14}+iu_{15} \\ \cdot & u_1 & u_{16}+iu_{17} & u_{18}+iu_{19} & u_{20}+iu_{21} & u_{22}+iu_{23} \\ \cdot & \cdot & u_2 & u_{24}+iu_{25} & u_{26}+iu_{27} & u_{28}+iu_{29} \\ \cdot & \cdot & \cdot & u_3 & u_{30}+iu_{31} & u_{32}+iu_{33} \\ \cdot & \cdot & \cdot & \cdot & u_4 & u_{34}+iu_{35} \\ \cdot & \cdot & \cdot & \cdot & \cdot & u_5 \end{pmatrix}. \tag{2}$$

  QUDA (column-major):

$$\begin{pmatrix} u_0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ u_6+iu_7 & u_1 & \cdot & \cdot & \cdot & \cdot \\ u_8+iu_9 & u_{16}+iu_{17} & u_2 & \cdot & \cdot & \cdot \\ u_{10}+iu_{11} & u_{18}+iu_{19} & u_{24}+iu_{25} & u_3 & \cdot & \cdot \\ u_{12}+iu_{13} & u_{20}+iu_{21} & u_{26}+iu_{27} & u_{30}+iu_{31} & u_4 & \cdot \\ u_{14}+iu_{15} & u_{22}+iu_{23} & u_{28}+iu_{29} & u_{32}+iu_{33} & u_{34}+iu_{35} & u_5 \end{pmatrix}. \tag{3}$$

Status

- ✓ Interface
- ✓ $C^\star$ boundaries
- QCD+QED Wilson-Clover

Status

- ✅ Interface
- ✅ $C^\star$ boundaries
- ✅ QCD+QED Wilson-Clover

Solver interface

- Solvers are called by means of their function, i.e. `cgne()`, `sap_gcr()`, `dfl_sap_gcr()`
- Usual utility:
  - input file parsing
  - solver setup
  - call solver

```
[Solver 0]
solver CGNE
nmx    256
res    1.0e-12
```

```
[Solver 1]
solver SAP_GCR
nkv    16
isolv  1
nmr    4
ncy    5
nmx    24
res    1.0e-8
```

```
[Solver 2]
solver DFL_SAP_GCR
idfl   0
nkv    16
isolv  1
nmr    4
ncy    5
nmx    24
res    1.0e-8
```

Figure: Example solver sections

Additional solver type

- Add solver type QUDA
- All options from `QudaInvertParam` and `QudaMultigridParam`

```
[Solver 3]
solver                    QUDA
gcrNkrylov                16
tol                       1e-12
inv_type                  QUDA_GCR_INVERTER
inv_type_precondition     QUDA_MG_INVERTER
...

[Solver 3 Multigrid]
n_level 2
...

[Solver 3 Multigrid Level 0]
...

[Solver 3 Multigrid Level 1]
...
```

Figure: Example QUDA solver section

Optimisations

- No doubling of the gauge field
- Calculate U(1) SW-term in QUDA (no transfer)
- Offload smearing, contractions
- Spinor field memory management (field unification)
- Partitioning
- multiple RHS

Performance

Tested system

- Tödi testing system at CSCS, Switzerland
- 4x NVIDIA® Grace™ CPU, 120GB RAM, 72 Neoverse V2 Armv9 cores
- 4x NVIDIA® H100 GPU, 96GB RAM
- NVLink® provides all-to-all cache-coherent memory between all host and device memory

Figure: Tödi: highest mountain in the Glarus Alps (3612 m)

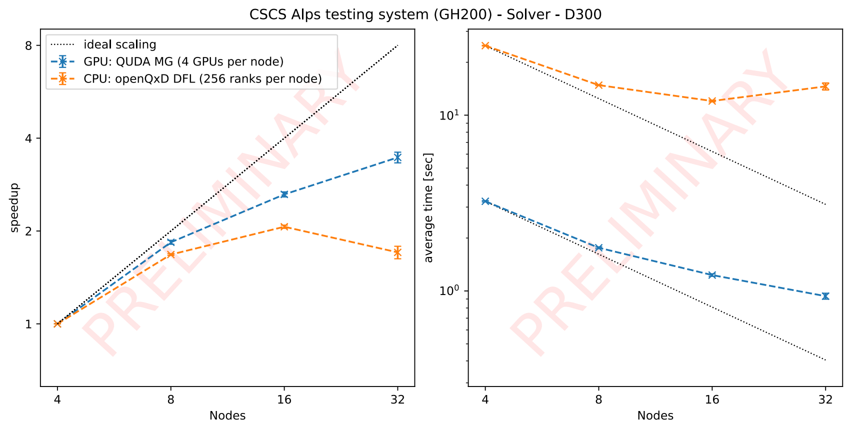

Figure: Strong scaling of one inversion of the Dirac operator; $T \times L^3 = 128 \times 64^3$, $m_\pi = 300$ MeV, C*-boundaries in all 3 spatial directions.

- GDR not yet available on Alps
- NVSHMEM not yet available on Alps

Conclusion

Conclusions

- Up and running interface to QUDA
- $C^\star$ boundaries in QUDA
- QCD+QED Wilson-Clover in QUDA
- Offloaded Dirac solves and eigensolver
- Contractions
- Smearing
- Field memory manager

Thanks for listening!

References I

[1] M. Lüscher et al., ***openQCD, simulation programs for lattice QCD***, (2012)

[2] M. Lüscher and S. Schaefer, **"Lattice QCD with open boundary conditions and twisted-mass reweighting"**, Comput. Phys. Commun. **184**, 519–528 (2013), arXiv:1206.2809 [hep-lat].

[3] A. S. Kronfeld and U. J. Wiese, **"SU(N) gauge theories with C-periodic boundary conditions (I). Topological structure"**, Nuclear Physics B **357**, 521–533 (1991).

[4] M. Lüscher, **"Local coherence and deflation of the low quark modes in lattice QCD"**, Journal of High Energy Physics **2007**, 081 (2007), eprint: 0706.2298,

[5] I. Campos et al., **"openQ*D code: a versatile tool for QCD+QED simulations"**, The European Physical Journal C **80**, 1–24 (2020), eprint: 1908.11673.

[6] R. Babich et al., **"Adaptive multigrid algorithm for the lattice Wilson-Dirac operator"**, Phys. Rev. Lett. **105**, 201602 (2010), arXiv:1005.3043 [hep-lat].

[7] J. Espinoza-Valverde, A. Frommer, G. Ramirez-Hidalgo, and M. Rottmann, **"Coarsest-level improvements in multigrid for lattice QCD on large-scale computers"**, Comput. Phys. Commun. **292**, 108869 (2023), arXiv:2205.09104 [math.NA].

References II

[8] M. A. Clark, R. Babich, K. Barros, R. C. Brower, and C. Rebbi, **"Solving lattice QCD systems of equations using mixed precision solvers on GPUs",** Computer Physics Communications **181**, 1517–1528 (2010), eprint: `0911.3191`.

Appendix

OPENQXD: SPACETIME ORDERING I

- txyz-convention, i.e. 4-vector $x = (x_0, x_1, x_2, x_3)$
- Lexicographical index (L_μ = rank-local lattice extent):

$$\Lambda(x, L) := L_3 L_2 L_1 x_0 + L_3 L_2 x_1 + L_3 x_2 + x_3. \tag{4}$$

- openQxD orders indices in cache-blocks: decomposition of the rank-local lattice into equal blocks of extent B_μ
  - Within a block: $\Lambda(b, B)$, where b = block-local Euclidean 4-vector
  - Block themselves: $\Lambda(n, N_B)$, where $N_{B,\mu} = L_\mu / B_\mu$ and $n_\mu = \lfloor x_\mu / B_\mu \rfloor$
- Even-odd ordering in the block (but not the blocks themselves)

$$\hat{x} = \left\lfloor \frac{1}{2} \Big(V_B \Lambda(n, N_B) + \Lambda(b, B) \Big) \right\rfloor + P(x) \frac{V}{2}, \tag{5}$$

where $V_B = B_0 B_1 B_2 B_3$ is the volume of a block, $P(x) = \frac{1}{2}(1 - (-1)^{\sum_\mu x_\mu})$ gives the parity and $V = L_3 L_2 L_1 L_0$.

OPENQXD: SPACETIME ORDERING II

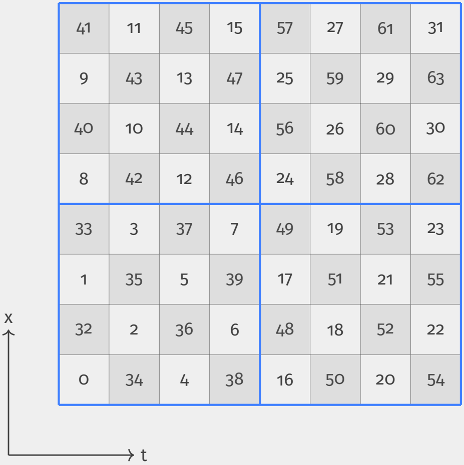

Figure: 2D example (8×8 local lattice) of the rank-local unique lattice index in openQxD (in time first convention (txyz)). The blue rectangles denote cache blocks of size 4×4. Gray sites are odd, white sites are even lattice points.

C⋆ Dirac operator I

The QCD+QED C⋆ Wilson-Clover Dirac operator in QCD simulations applied onto a spinor field $\psi(x)$ is (the lattice spacing is set to $a = 1$)

$$
\begin{aligned}
D_{\mathrm{w}}\psi(x) &= (4 + m_0)\psi(x) \\
&-\frac{1}{2}\sum_{\mu=0}^{3}\left\{H_\mu(x)(1-\gamma_\mu)\psi(x+\hat{\mu}) + H_\mu(x-\hat{\mu})^{-1}(1+\gamma_\mu)\psi(x-\hat{\mu})\right\} \\
&+c_{\mathrm{sw}}^{SU(3)}\frac{i}{4}\sum_{\mu,\nu=0}^{3}\sigma_{\mu\nu}\hat{F}_{\mu\nu}(x)\psi(x) + qc_{\mathrm{sw}}^{U(1)}\frac{i}{4}\sum_{\mu,\nu=0}^{3}\sigma_{\mu\nu}\hat{A}_{\mu\nu}\psi(x)\,,
\end{aligned} \tag{6}
$$

where the gauge field $H_\mu(x)$ is the $U(3)$-valued link between extended lattice point x and $x+\hat{\mu}$, the γ_μ are the Dirac matrices obeying the Euclidean Clifford algebra, $\{\gamma_\mu, \gamma_\nu\} = 2\delta_{\mu\nu}$ and $\sigma_{\mu\nu} = \frac{i}{2}[\gamma_\mu, \gamma_\nu]$.

$C^\star$ Dirac operator II

The $SU(3)$ field strength tensor $\hat{F}$ is defined as

$$
\begin{aligned}
\hat{F}_{\mu\nu}(x) &= \frac{1}{8}\left\{Q_{\mu\nu}(x) - Q_{\nu\mu}(x)\right\}, \\
Q_{\mu\nu}(x) &= U_\mu(x)U_\nu(x+\hat{\mu})U_\mu(x+\hat{\nu})^{-1}U_\nu(x)^{-1} \\
&+ U_\nu(x)U_\mu(x-\hat{\mu}+\hat{\nu})^{-1}U_\nu(x-\hat{\mu})^{-1}U_\mu(x-\hat{\mu}) \\
&+ U_\mu(x-\hat{\mu})^{-1}U_\nu(x-\hat{\mu}-\hat{\nu})^{-1}U_\mu(x-\hat{\mu}-\hat{\nu})U_\nu(x-\hat{\nu}) \\
&+ U_\nu(x-\hat{\nu})^{-1}U_\mu(x-\hat{\nu})U_\nu(x+\hat{\mu}-\hat{\nu})U_\mu(x)^{-1}
\end{aligned}
$$

where the gauge field $U_\mu(x)$ is $SU(3)$-valued

$C^\star$ Dirac operator III

We add the $U(1)$ SW-term,

$$D_{\mathrm{w}} \to D_{\mathrm{w}} + q c_{\mathrm{sw}}^{U(1)} \frac{i}{4} \sum_{\mu,\nu=0}^{3} \sigma_{\mu\nu} \hat{A}_{\mu\nu} \,, \tag{7}$$

where q is the charge and the $U(1)$ field strength tensor $\hat{A}_{\mu\nu}(x)$ is defined as

$$\hat{A}_{\mu\nu}(x) = \frac{i}{4q_{el}} \mathrm{Im}\left\{ z_{\mu\nu}(x) + z_{\mu\nu}(x-\hat{\mu}) + z_{\mu\nu}(x-\hat{\nu}) + z_{\mu\nu}(x-\hat{\mu}-\hat{\nu}) \right\}$$

$$z_{\mu\nu}(x) = e^{i\{A_\mu(x)+A_\nu(x+\hat{\mu})-A_\mu(x+\hat{\nu})-A_\nu(x)\}}$$

$C^\star$ BOUNDARY CONDITIONS

The implementation of the $C^\star$ boundary conditions for the fields is the following (orbifold construction):

$$
\begin{aligned}
A_\mu(x + L_k\hat{k}) &= -A_\mu,\\
\psi_f(x + L_k\hat{k}) &= C^{-1}\overline{\psi}_f^T(x),\\
\overline{\psi}_f(x + L_k\hat{k}) &= -\psi_f^T(x)C,\\
U_\mu(x + L_k\hat{k}) &= U^*\mu(x),
\end{aligned}
\tag{8}
$$

where L_k is the size of the lattice in direction $\hat{k}$, U^* denotes complex conjugation. The charge–conjugation matrix C satisfies

$$
C^T = -C, \quad C^\dagger = C^{-1}, \quad C^{-1}\gamma_\mu C = -\gamma_\mu^T. \tag{9}
$$

The gauge action is

$$
S_{g,SU(3)} = \frac{1}{g_0^2}\sum_{\mathcal{C}\in\mathcal{S}_0} \mathrm{tr}\left[1 - U(\mathcal{C})\right], \tag{10}
$$

$$
S_{g,U(1)} = \frac{1}{2q_{el}^2 e_0^2}\sum_{\mathcal{C}\in\mathcal{S}_0} \mathrm{tr}\left[1 - z(\mathcal{C})\right], \tag{11}
$$

where the bare coupling constants are $g_0, e_0, q_{el} = 1/6$. Given a path $\mathcal{C}$ on a lattice, $U(\mathcal{C})$ and $Z(\mathcal{C})$ denote the $SU(3)$ and $U(1)$ parallel transport along $\mathcal{C}$.

Why the doubled lattice?

- On the extended lattice, points x and $x + L_k\hat{k}$ do not coincide!
- Admissible fields are given by the boundary conditions
- Admissible gauge fields on mirror lattice are completely determined by their value on the physical lattice
- On physical lattice: ψ and $\bar{\psi}$ are independent Grassmann variables
- On extended lattice: $\bar{\psi}$ is completely determined by ψ
- Integration measure for fermion field:

$$[\mathrm{d}\psi]_{\Lambda_{phys}} [\mathrm{d}\bar{\psi}]_{\Lambda_{phys}} = \prod_{x\in\Lambda_{phys}} \mathrm{d}\psi(x)\bar{\psi}(x) = \prod_{x\in\Lambda_{exended}} \mathrm{d}\psi(x) = [\mathrm{d}\psi]_{\Lambda_{extended}} \tag{12}$$

$\Longrightarrow$ We need the doubled lattice for the fermion field!

DIRAC OPERATOR SCALING I

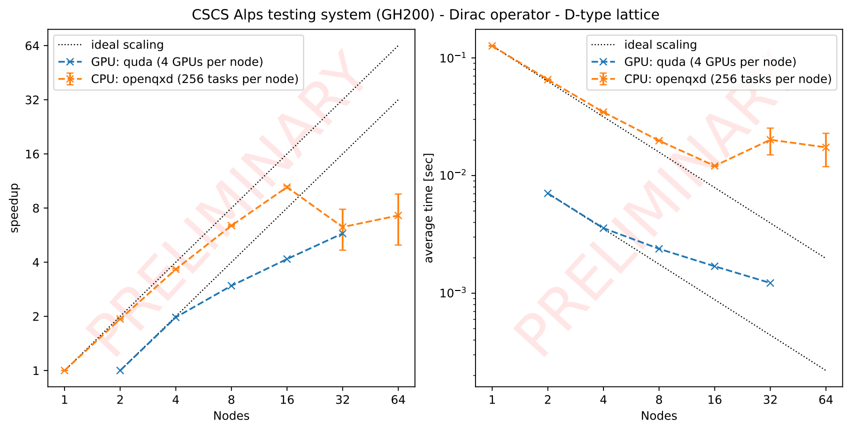

Figure: C* Wilson-Clover Dirac operator strong scaling

- GDR not yet available on Alps
- NVSHMEM not yet available on Alps

Dirac operator scaling II

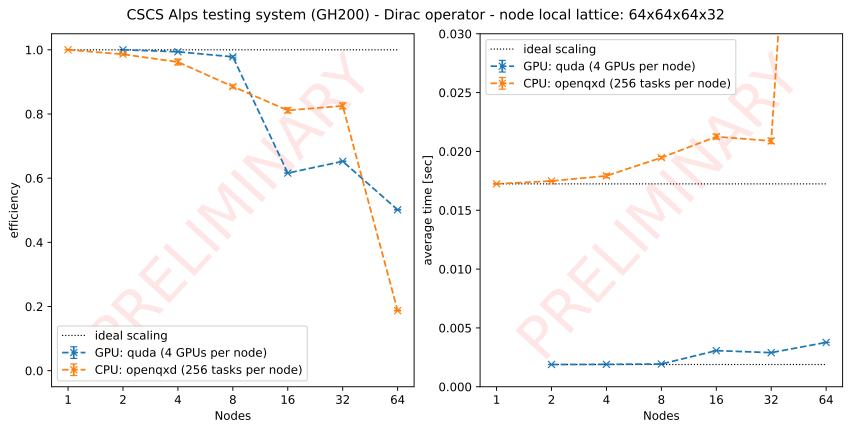

Figure: C⋆ Wilson-Clover Dirac operator weak scaling

- GDR not yet available on Alps
- NVSHMEM not yet available on Alps

Unification of fields

MOTIVATION

- Initial code: all functions implemented in CPU $\rightarrow$ no transfers needed
- Ideal final code: all functions implemented in GPU $\rightarrow$ no transfers needed $\longrightarrow$ we'll probably never reach that
- Intermediate phase: some functions are ported to GPU, but not all of them $\rightarrow$ needs transfers

Motivation

- Initial code: all functions implemented in CPU $\rightarrow$ no transfers needed
- Ideal final code: all functions implemented in GPU $\rightarrow$ no transfers needed $\longrightarrow$ we'll probably never reach that
- Intermediate phase: some functions are ported to GPU, but not all of them $\rightarrow$ needs transfers

Requirement 1

We don't want to rewrite every program, when a new function is ported to GPU!

Motivation

- Initial code: all functions implemented in CPU $\rightarrow$ no transfers needed
- Ideal final code: all functions implemented in GPU $\rightarrow$ no transfers needed $\longrightarrow$ we'll probably never reach that
- Intermediate phase: some functions are ported to GPU, but not all of them $\rightarrow$ needs transfers

Requirement 1

We don't want to rewrite every program, when a new function is ported to GPU!

Requirement 2

Fully backwards compatible with openQxD's memory layout

OPENQXD: OVERLOADING OF FUNCTIONS I

```
#if (defined AVX)
// implementation using AVX intrinsics
void functionA(spinor_dble *s) { ... }
#elif (defined x64)
// implementation using SSE2 intrinsics
void functionA(spinor_dble *s) { ... }
#else
// default implementation
void functionA(spinor_dble *s) { ... }
#endif
```

Figure: Example overloading of `functionA`.

OPENQXD: OVERLOADING OF FUNCTIONS II

```c
#if (defined AVX)
// implementation using AVX intrinsics
void functionA(spinor_dble *s) { ... }
#elif (defined x64)
// implementation using SSE2 intrinsics
void functionA(spinor_dble *s) { ... }
#elif (defined GPU_OFFLOADING)
// GPU overloading of the function
void functionA(spinor_dble *s) { ... }
#else
// default implementation
void functionA(spinor_dble *s) { ... }
#endif
```

Figure: Example overloading of `functionA`.

Unified fields

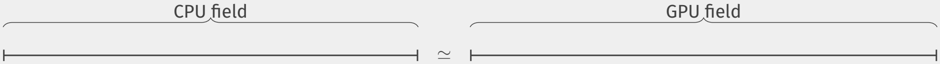

Figure: Each field with openQxD corresponds to a field within QUDA.

- openQxD operates on base pointers of struct-arrays
- Establish a 1-1 correspondence between CPU/GPU fields

$\Longrightarrow$ Everytime (de-)allocating a field $\rightarrow$ (de-)allocate on both devices

$\Longrightarrow$ Maintain consistency among the two fields (CPU/GPU manipulates field)

Maintaining consistency

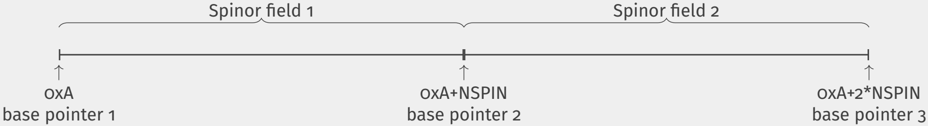

Figure: Current field allocation scheme.

Maintaining consistency

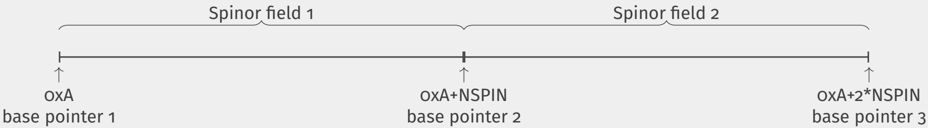

Figure: Current field allocation scheme.

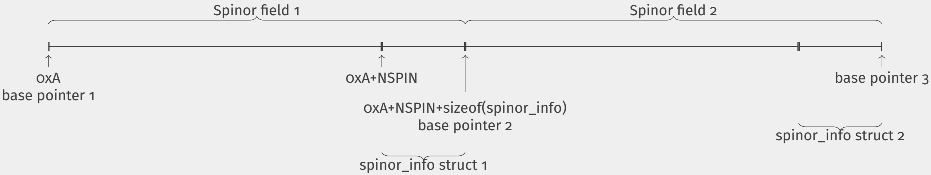

Figure: New field allocation scheme (`spinor_info` struct *after* the data).

spinor_info STRUCT

Information held by the `spinor_info` struct:

- Field status: `CPU_NEWER`, `GPU_NEWER`, `IN_SYNC`
- GPU pointer: pointer to field on the GPU (i.e. pointer to `ColorSpinorField` instance)
- Other information: eg. field size in bytes, stats, ...

$\Longrightarrow$ Only changes in the (de-)allocation functions: `alloc_wsd( )`, `reserve_wsd( )`, `release_wsd( )` + their single precision variants

Procedure

- Functions within openQxD still operate on base pointers (in the same way as before!) $\implies$ they all still work (no change needed)
- GPU-offloaded functions now take the same CPU base pointer
  1. Navigate to the `spinor_info` struct
  2. Check if field needs to be transferred
  3. Transfer if needed
  4. Obtain GPU field pointer from info struct
  5. Update status field in info struct
  6. Continue function body with GPU field
- openQxD functions take the usual CPU base pointer
  1. Navigate to the `spinor_info` struct
  2. Check if field needs to be transferred
  3. Transfer if needed
  4. Update status field in info struct
  5. Continue function body with CPU field