Multigrid for Wilson Clover Fermions in Grid

Daniel Richtmann, Tilo Wettig, Peter Boyle

July 23rd, 2018

DISCLAIMER

Everything I say about Grid here is my opinion only.

OUTLINE

Motivation

Grid

Implementation details

Performance

Comparison with DDalphaAMG

Conclusions

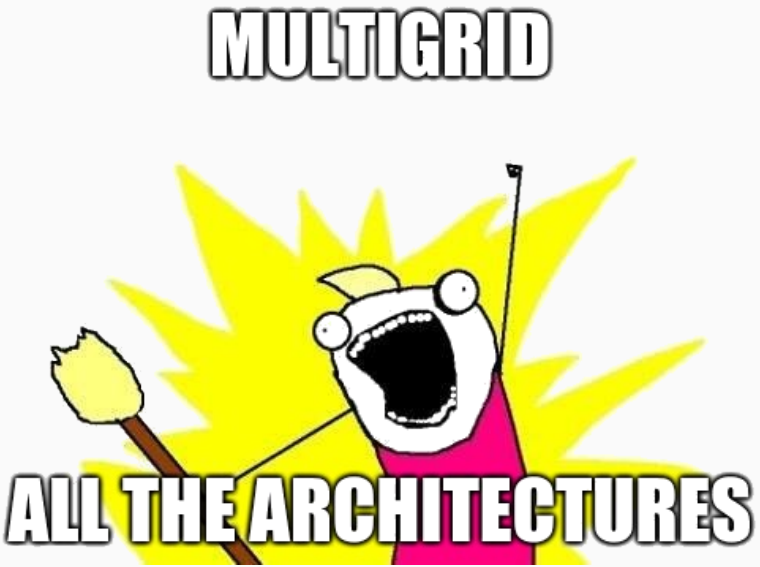
Motivation

Multigrid by now default solver algorithm for Wilson Clover fermions
Different architectures, different repositories

- DDalphaAMG library by Matthias Rottmann et al. (SSE only) [1303.1377], GitHub
- Xeon Phi implementation of DDalphaAMG by RQCD [1512.04506], [1710.07041]
- Chroma MG
- QUDA MG by Kate Clark et al. [0911.3191], GitHub
- New architectures? (see poster by Nils Meyer)

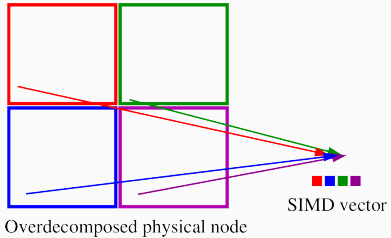
Plan: Unify in single implementation and have well-performing solver on all non-GPU architectures

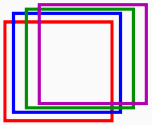
$\rightarrow$ Grid [1512.03487], GitHub


MULTIGRID
ALL THE ARCHITECTURES

Grid

- Data-parallel library (just like QDP++)
- Elegant high-level interface to QCD due to nested tensors + slim ET engine
- Satisfies all three major parallelism paradigms of CPUs
  - SIMD (big focus on site-fusing)
  - Threading with OpenMP (fork join model)
  - Message passing (MPI)
- User/Developer experience: QCD toolbox with lots of good stuff present
  - Almost like writing python/matlab code
  - Allows for rapid development





Basic building blocks for 2-level MG present [1611.06944]

```
template<class Fobj, class CComplex, int nbasis>
class Aggregation {
public:
  // ...
  Aggregation(GridBase *_CoarseGrid, GridBase *_FineGrid,
              int _checkerboard);
  void Orthogonalise();
  void CheckOrthogonal();
  void CreateSubspaceRandom(GridParallelRNG &RNG);
  void CreateSubspace(GridParallelRNG &RNG,
                      LinearOperatorBase<FineField> &hermop,
                      int nn = nbasis);
  void ProjectToSubspace(CoarseVector &CoarseVec,
                         const FineField &FineVec);
  void PromoteFromSubspace(const CoarseVector &CoarseVec,
                           FineField &FineVec);
};
```

```
template<class Fobj, class CComplex, int nbasis>
class CoarsenedMatrix :
public SparseMatrixBase<Lattice<iVector<CComplex, nbasis>>> {
public:
  // ...
  CoarsenedMatrix(GridCartesian &CoarseGrid);
  RealD M(const CoarseVector &in, CoarseVector &out);
  RealD Mdag(const CoarseVector &in, CoarseVector &out);
  void  Mdiag(const CoarseVector &in, CoarseVector &out);
  void  Mdir(const CoarseVector &in, CoarseVector &out,
             int dir, int disp);
  void  CoarsenOperator(GridBase *FineGrid,
                        LinearOperatorBase<Lattice<Fobj>> &linop,
                        Aggregation<Fobj, CComplex, nbasis> &Subspace);
};
```

But: No support for further coarsening. Requires from `CoarsenedMatrix`

- `Mdiag`
- `Mdir`

Implementation details

IMPLEMENTATION DETAILS – OVERVIEW

- Algorithmic goal: DDalphaAMG + variations
- Multilevel MG by enabling `CoarsenedMatrix` to be coarsened ✓
- Wuppertal iterative setup phase ✓
- Choice between GMRES/MR as smoother ✓
- Want to run MG preconditioner in lower precision than outer solver
  - ✓ Single precision MG / double precision outer solver trivial in Grid (`precisionChange`)
  - ? Half precision MG / double precision outer solver (no HP compute support)
- TODOs
  - Schwarz smoothing
  - Red-black preconditioning of coarse solver
  - Interface for simulation programs (Hadrons, Chroma, ...)

- MG requirement: need to preserve γ_5-hermiticity of D/chirality on coarser grids
- Grid's coarsening ecosystem is not (yet?) aware of internal dofs of a site spinor

```
typedef Lattice<iScalar<iVector<iVector<vComplex, Nc>, Ns> > >           FineVector;
typedef Lattice<iVector<iScalar<iScalar<vComplex> > >, nbasis> > CoarseVector;
```

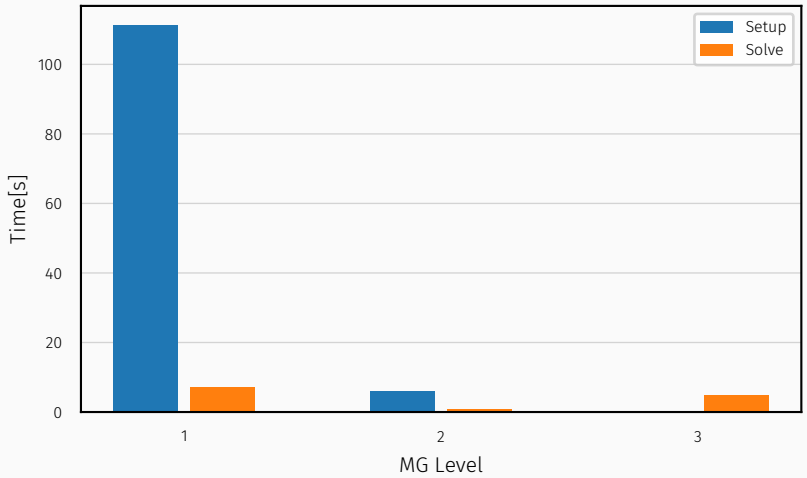
  ⟶ Forces us to do explicit chiral doubling of the null-space vectors, i.e., create $2N$ vectors out of N by using chiral projectors

$$v_i = \frac{1+\gamma_5}{2} v_{0,i}, \quad v_{i+N} = \frac{1-\gamma_5}{2} v_{0,i}$$

- Doubles memory requirement for null-space vectors w.r.t. DDalphaAMG for same coarse-grid size + means more work in setup
- Something in the pipeline ⟶ WIP

Performance

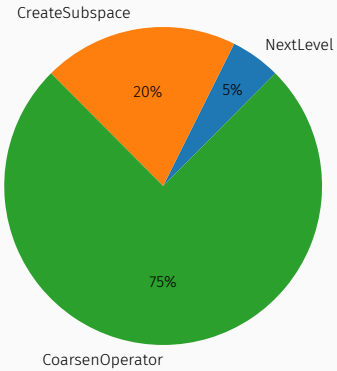
Runtime distribution using DDalphaAMG default parameters



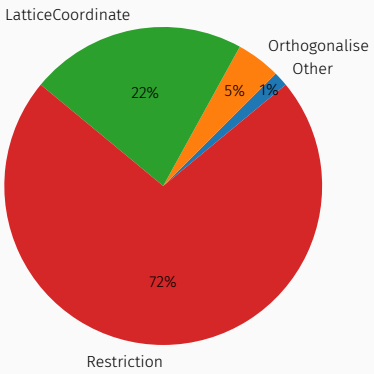
Clearly dominated by setup on finest level

Runtime distribution using DDalphaAMG default parameters: Setup

Setup



Setup – CoarsenOperator



Note: This is vanilla Grid code

PERFORMANCE – RESTRICTION OPERATOR

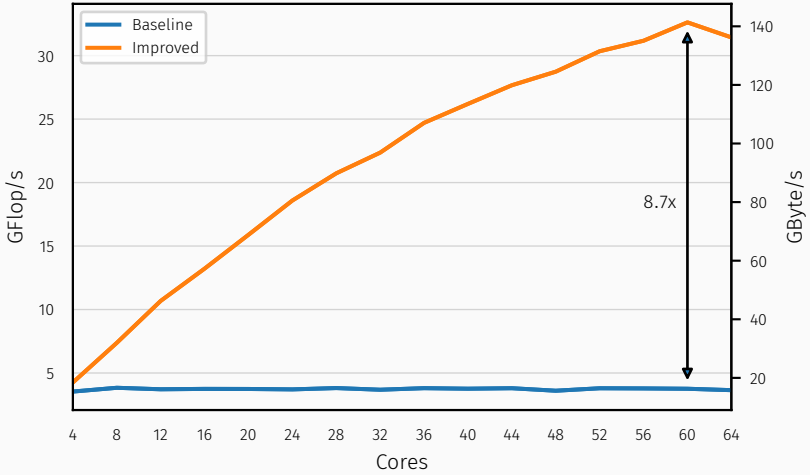
```
parallel_for(int sf=0; sf<fine->oSites(); sf++){
  // ...
  PARALLEL_CRITICAL
  for(int i=0; i<nbasis; i++)
    coarseData._odata[sc](i) = coarseData._odata[sc](i)
    + innerProduct(Basis[i]._odata[sf], fineData._odata[sf]);
  // ...
}
```

Baseline: Critical region is the problem, code basically runs serial

```
// ...
parallel_for(int sc=0; sc<coarse->oSites(); sc++) {
  for(auto sf : lookUpTable[sc]) {
    for(int i=0; i<nbasis; i++) {
      coarseData._odata[sc](i) = coarseData._odata[sc](i)
      + innerProduct(Basis[i]._odata[sf], fineData._odata[sf]);
    }
  }
}
```

Improvement: Do index calculation serially and thread calculation over coarse sites, not fine

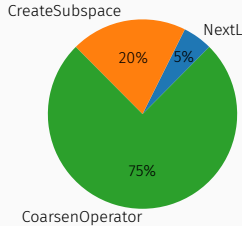
Implications on performance

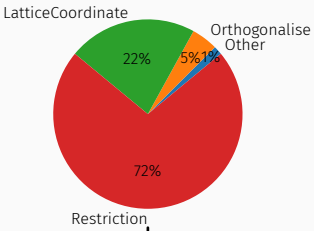


Lack of sustained memory bandwidth stems from irregular access pattern → WIP

PERFORMANCE – SETUP AFTER IMPROVEMENT OF RESTRICTION

Before



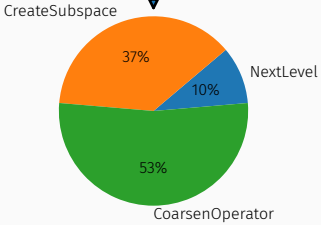


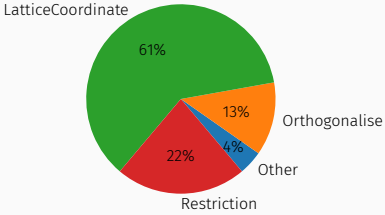
2.1x

Total MG setup:
2.0x

2.8x

After





Comparison with DDalphaAMG

A FIRST COMPARISON WITH DDALPHAAMG – TEST SETUP

- Fix GMRES as smoother
- DDalphaAMG supports vectorization with SSE, but only when even-odd preconditioning is on
- Grid MG supports all common SIMD extensions, but can't do even-odd preconditioning yet
- Could turn off SIMD for both, pointless
- Decided to let DDalphaAMG use SSE + even-odd and Grid MG use AVX
- Test setup (other parameters are DDalphaAMG defaults):

System	Lattice	Blocksize	Mass	Basis vectors	Smoother
8-core Broadwell	16^4	4^4	-0.25	20	GMRES

A FIRST COMPARISON WITH DDALPHAAMG – RESULTS

Setup iter	Outer Iter		Init Setup		Iter Setup		Solve		Total	
Repo	D	G	D	G	D	G	D	G	D	G
0	84	82	9.7	39.4	0	0	18.1	14.2	37	53
1	74	71	9.8	39.4	8.5	32.3	17.4	20.3	35	92
2	34	53	10.0	38.9	16.8	62.8	16.8	16.7	43	117
3	21	22	9.9	40.0	29.4	92.9	5.9	3.8	45	136
4	20	21	9.7	39.9	38.8	121	6.1	3.5	54	164
5	20	20	9.8	39.5	47.7	158	5.8	3.7	63	201
6	20	21	9.2	39.1	55.9	193	6.2	5.0	71	237
7	20	20	9.8	39.7	64.2	226	6.3	3.3	80	269
8	20	20	9.7	39.9	72.2	251	6.1	3.2	88	294
9	20	20	9.9	40.0	80.6	286	6.3	3.3	96	329
10	20	20	9.6	39.8	86.8	316	6.0	3.5	102	359

Conclusions

CONCLUSIONS & OUTLOOK

- Multilevel MG solver for Wilson Clover now present in Grid GitHub, PR pending
- But: Has some performance pitfalls that need to be resolved → WIP
- First comparison to DDalphaAMG: Solve time as expected, losing in setup
- Future directions:
  - Improve upon setup
  - Implement missing parts (Schwarz, Even-odd, ...)
  - Interface for simulation programs
  - Investigate multi-node behavior