CL\textsuperscript{2}QCD

Developers

♂ Alessandro Sciarra
♂ Christopher Czaban
♀ Francesca Cuteri

Past Developers

♂ Matthias Bach
♂ Christopher Pinke
♂ Frederik Depta
♂ Christian Schäfer
♂ Lars Zeidlewicz

ZEUHEN

F RANKFURT

https://github.com/CL2QCD/cl2qcd

Lattice Seminar
20.06.2016
DESY

Lattice group of Owe Philipsen
Our physics motivation (I)

\[ N_s^3 \times N_t \text{ lattice} \rightarrow T = \frac{1}{a N_t} \rightarrow 1 \ll N_t \ll N_s \]

QCD THERMODYNAMICS

Second order scenario

First order scenario
Our physics motivation (II)

In the volume below the plane $\mu = 0$ simulations are safe.

The $A$ point in the Roberge–Weiss plane is at a value of the mass that can be simulated.

The position of the point $B$ determines the type of transition at the $N_f = 2$ chiral point at $\mu = 0$.

The idea is to study the line connecting $A$ and $B$ in the $m_s = \infty$ plane.

Why do we need a fast code?

Just to give an idea of how costly it is...

We keep fixed: \( N_t \) \( \mu \)

```bash
for mass in ...; do      # ~6 values
 for Ns in ...; do      # ~3 values >= 3*Nt
  for T in ...; do      # ~5 values
    echo "Run the (R)HMC for >50k trajectories"
    # ...
    done
  done
done
```

Lattice Seminar
DESY
Alessandro Sciarra

Motivation and Philosophy

CL\textsuperscript{2}QCD

HIC for FAIR – 20.06.2016
Why do we need a fast code?

Just to give an idea of how costly it is...

We keep fixed: $N_t$

```bash
for mu in ...; do # ~6 values
    for mass in ...; do # ~6 values
        for Ns in ...; do # ~3 values >= 3*Nt
            for T in ...; do # ~5 values
                echo "Run the (R)HMC for >50k trajectories"
                # ...
            done
        done
    done
done
done
```
Why do we need a fast code?

Just to give an idea of how costly it is...

```bash
for Nt in ...; do # ~3 values
   for mu in ...; do # ~6 values
      for mass in ...; do # ~6 values
         for Ns in ...; do # ~3 values >= 3*Nt
            for T in ...; do # ~5 values
               echo "Run the (R)HMC for >50k trajectories"
            # ...
            done
         done
      done
   done
done
   # Consider that the typical time of a
   done # simulation varies from weeks to months
done
```
Why do we need a fast code?

Just to give an idea of how costly it is...

```bash
for Nt in ...; do # ~3 values
    for mu in ...; do # ~6 values
        for mass in ...; do # ~6 values
            for Ns in ...; do # ~3 values >= 3*Nt
                for T in ...; do # ~5 values
                    echo "Run the (R)HMC for >50k trajectories"
                    # ...
                    done
                done
            done # Consider that the typical time of a
            done # simulation varies from weeks to months
        done
    done
done
```

Considering 3 months as average time of a single simulation…

- For loop in order as above → 405 years (roughly 1.5)
- Inner three for loop parallel → 4.5 years

(STRETCH FACTOR)

(feedback, technical)
Should a code *just* work?

Any code in principle should be:
- Readable
- Maintainable
- Easy to extend
- Easy to use
- Hard to break
- Testable

Clean Code 2 Limit Questions and Doubts

Alessandro Sciarra (Goethe Universität)
Should a code *just* work? *NO*

Any code in principle should be:

- Readable
- Maintainable
- Easy to extend
- Easy to use
- Hard to break
- Testable
Any code in principle should be:

- Readable
- Maintainable
- Easy to extend
- Easy to use
- Hard to break
- Testable

Should a code just work? NO
Should a code *just* work?  

**NO**

Any code in principle should be:

- **Readable**
- **Maintainable**
- **Easy to extend**
- **Easy to use**
- **Hard to break**
- **Testable**

---

**CL$^2$QCD**

**ALWAYS CODE AS IF THE GUY WHO ENDS UP MAINTAINING YOUR CODE WILL BE A VIOLENT PSYCHOPATH WHO KNOWS WHERE YOU LIVE.**
Should a code just work?

NO

Any code in principle should be:

- Readable
- Maintainable
- Easy to extend
- Easy to use
- Hard to break
- Testable

Lattice QCD with Open CL

CL²QCD

ALWAYS CODE AS IF THE GUY WHO ENDS UP MAINTAINING YOUR CODE WILL BE A VIOLENT PSYCHOPATH WHO KNOWS WHERE YOU LIVE.
Any code in principle should be:

- Readable
- Maintainable
- Easy to extend
- Easy to use
- Hard to break
- Testable

CL\textsuperscript{2}QCD

Lattice QCD with Open CL

Clean Code 2 Limit Questions and Doubts
Outline of the talk

1. Motivation and our philosophy

2. CL$^2$QCD features

3. Structure of the code
   - Physics
   - Hardware

4. Programming on GPU

5. Unit tests, maintainability and portability

6. Performances of the code
   - Multiple GPUs

7. Ongoing and future work
Outline of the talk

1. Motivation and our philosophy

2. CL²QCD features

3. Structure of the code
   - Physics
   - Hardware

4. Programming on GPU

5. Unit tests, maintainability and portability

6. Performances of the code
   - Multiple GPUs

7. Ongoing and future work
The code from the user point of view (I)

Features implemented in CL²QCD

- Different fermion formulations
  - Wilson fermions in standard formulation
  - Staggered fermions in rooted standard formulation
  - Twisted Mass Wilson fermions

- Improved gauge actions (for Monte Carlo simulations)
  - Tree-level Symanzik
  - Iwasaki
  - Doubly Blocked Wilson

- Pure gauge simulations (Heatbath and Monte Carlo)

- Standard inversion and integration algorithms

- ILDG-compatible input/output (using LIME)

- RANLUX pseudo-random number generator

- Even-odd preconditioning

- Hasenbuch’s trick for Wilson fermions
The code from the user point of view (II)

**Installation procedure**

```
# Clone the git repository
cd <CL2QCD_INSTALL_DIR>
git clone https://github.com/CL2QCD/cl2qcd.git
# Make a build directory
cd <CL2QCD_INSTALL_DIR>/cl2qcd
mkdir build
cd build/
#
# Run cmake; if not found automatically,
# cmake variables can be set via command line as
# "-D <CMAKE_VARIABLE>=<VALUE>"
#
cmake ..
# Build all executables
make -j
```

The compiler must be capable of basic C++11 features

**Required libraries**

- **OpenCL** → http://www.khronos.org/opencl/
- **LIME** → http://usqcd.jlab.org/usqcd-docs/c-lime/
- **libxml2** → http://xmlsoft.org/
- **Boost** → http://www.boost.org/
- **GMP** → http://gmplib.org/
- **MPFR** → http://www.mpfr.org/
- **Nettle** → http://www.lysator.liu.se/~nisse/nettle/
The code from the user point of view (III)

**EXECUTABLES**

- **HMC**: Only with (Twisted Mass) Wilson fermion.
- **SU(3) Heatbath**: Possibility of doing overrelaxation.
- **RHMC**: Only with standard staggered fermion.
- **Inverter**: Measurement of fermionic observables.
- **Gauge observables**: Measurement of gauge observables.

**CL²QCD features**

- Structure of the code
- Physics
- Hardware
- Programming on GPU
- Unit tests, maintainability and portability
- Performances of the code
- Multiple GPUs
- Ongoing and future work
Executable Options

 EACH executable has its helper → ./<executable> -h
 EACH option has a default value, used if not specified
 !!! In the helpers there are options not relevant for that algorithm
 ➤ Some options are used for profiling/benchmarks only

Possible input file:

```plaintext
... HMC_options:

--tau arg (=0.5)
--reversibility_check arg (=0)
--integrationsteps0 arg (=10)
--integrationsteps1 arg (=10)
--integrationsteps2 arg (=10)
--hmcsteps arg (=10)
--num_timescales arg (=1)
--integrator0 arg (=leapfrog)
--integrator1 arg (=leapfrog)
--integrator2 arg (=leapfrog)
--lambda0 arg (=0.19318332750378359)
--lambda1 arg (=0.19318332750378359)
--lambda2 arg (=0.19318332750378359)
--use_gauge_only arg (=0)
--use_mp arg (=0)
...
```

...
The output levels of the code

Einhard

© 2010 Matthias Bach
The output levels of the code

OFF

FATAL

ERROR

WARN

INFO

DEBUG

TRACE

[01:05:26] FATAL:
[22:13:59] ERROR:
[15:42:13] WARNING:
[10:00:12] INFO:
[03:35:17] DEBUG:
[08:52:48] TRACE:

The undesired output is removed by pre-processor operations.

NO OVERHEAD
The output levels of the code

- **OFF**
- **FATAL**
- **ERROR**
- **WARN**
- **INFO**
- **DEBUG**
- **TRACE**

*Einhard* © 2010 Matthias Bach

- [01:05:26] FATAL:
- [22:13:59] ERROR:
- [15:42:13] WARNING:
- [10:00:12] INFO:
- [03:35:17] DEBUG:
- [08:52:48] TRACE:

The undesired output is removed by pre-processor operations

**NO OVERHEAD**
Outline of the talk

1. Motivation and our philosophy

2. CL^2QCD features

3. Structure of the code
   - Physics
   - Hardware

4. Programming on GPU

5. Unit tests, maintainability and portability

6. Performances of the code
   - Multiple GPUs

7. Ongoing and future work
The code structure

M. Bach, O. Philipsen, C. Pinke, and A. Sciarra «CL²QCD– Lattice QCD based on OpenCL»

http://publikationen.ub.uni-frankfurt.de/frontdoor/index/index/docId/37074
The code structure

CL²QCD features
Structure of the code

Physics
Hardware
Programming on GPU
Unit tests, maintainability and portability
Performances of the code
Multiple GPUs
Ongoing and future work
The code structure

**CL\(^2\)QCD**

**PHYSICS**

**META**

**HARDWARE**

**OPEN CL Kernels**

**CL\(^2\)QCD features**

- Physics
- Hardware
- Programming on GPU
- Unit tests, maintainability and portability
- Performances of the code
- Multiple GPUs
- Ongoing and future work
The code structure

- **Physics**
  - Algorithms
  - Fermion matrices
  - Lattice fields
  - PRNG
  - Noise Sources
  - Observable

- **Meta**
  - Parameters
  - ILDG I/O
  - Utilities

- **Hardware**
  - System
  - Device
  - Code
  - Open CL compiler
  - Lattice Buffers

- **Open CL Kernels**

---

**Structure of the code**

- CL$^2$QCD features
- Physics
- Hardware
- Programming on GPU
- Unit tests, maintainability and portability
- Performances of the code
- Multiple GPUs
- Ongoing and future work
Outline of the talk

1. Motivation and our philosophy

2. CL$^2$QCD features

3. Structure of the code
   - Physics
   - Hardware

4. Programming on GPU

5. Unit tests, maintainability and portability

6. Performances of the code
   - Multiple GPUs

7. Ongoing and future work
The Physics package

Lattice Seminar
DESY
ALESSANDRO SCIARRA

CL^2QCD features
Structure of the code

Physics
Hardware
Programming on GPU
Unit tests, maintainability and portability
Performances of the code
Multiple GPUs
Ongoing and future work

Physic

ψ-fields
U-µ-fields
H-µ-fields
F-fermions
Wilson
TwistedMass
Staggered
Gauge
OBS.
Plaquette
PolyakovLoop
ChiralCondensate
ALGS
Integrator
Leapfrog
2MN
Fermionforce
Solver
CG
CG-M
BiCGstab
PRNG
NOISE
SOURCES
Point
Slices
Volume
TESTS

Alessandro Sciarra (Goethe Universität)
The Physics package

- **Gauge Obs.**
- **Fermion Matrices**
- **ALGS**
- **PRNG**
- **Noise Sources**
- **Lattice Fields**

**Tests**
The Physics package

- Polyakov Loop
- Plaquette
- Staggered
- Twisted Mass
- Wilson
- H_{\mu}\text{-fields}
- U_{\mu}\text{-fields}
- \psi\text{-fields}

Gauge Observables

- Chiral Condensate
- \Psi\text{-fields}
- U_{\mu}\text{-fields}
- \psi\text{-fields}

Fermion Matrices

- Fermion
- Integrator
- ALGS
- Solver

Physics Algorithms

- PRNG
- Noise Sources
- Point
- Volume
- Slices

Tests

Integration

- Wilson

Lattice Fields

- Fermion force
- Solver

Physics Sources

- Ongoing and future work
The Physics package

Lattice Seminar
DESY

LESSANDRO SCIARRA

CL²QCD features
Structure of the code

Physics
Hardware
Programming on GPU
Unit tests, maintainability and portability
Performances of the code
Multiple GPUs
Ongoing and future work

The Physics package

CL²QCD

The Physics package

GAUGE OBS.

FERMION MATRICES

GAUGE OBS.

PLAQUETTE

Staggered

Twisted Mass

Wilson

H_{\mu}\text{-fields}

U_{\mu}\text{-fields}

\psi\text{-fields}

LATTICE FIELDS

NOISE SOURCES

Test:

ALGS

Integrator

Leapfrog

2MN

Fermion force

Solver

CG

CG-M

BiCGstab

PRNG

PGM

Polyakov Loop

Chiral Condensate

\psi\text{-fields}

\U_{\mu}\text{-fields}

Volume

Slices

Point

TESTS

Wilson

H_{\mu}\text{-fields}

U_{\mu}\text{-fields}

\psi\text{-fields}

LATTICE FIELDS

NOISE SOURCES

Test:

ALGS

Integrator

Leapfrog

2MN

Fermion force

Solver

CG

CG-M

BiCGstab

PRNG

PGM

Polyakov Loop

Chiral Condensate

\psi\text{-fields}

\U_{\mu}\text{-fields}

Volume

Slices

Point

TESTS

Wilson

H_{\mu}\text{-fields}

U_{\mu}\text{-fields}

\psi\text{-fields}

LATTICE FIELDS

NOISE SOURCES

Test:
Outline of the talk

1. Motivation and our philosophy
2. CL²QCD features
3. Structure of the code
   - Physics
   - Hardware
4. Programming on GPU
5. Unit tests, maintainability and portability
6. Performances of the code
   - Multiple GPUs
7. Ongoing and future work
The Hardware package

Hardware

Buffers

\( \psi \)-fields

\( U_\mu \)-fields

\( H_\mu \)-fields

Ongoing and future work

- Programming on GPU
- Unit tests, maintainability and portability
- Performances of the code
- Multiple GPUs

Ongoing and future work

- Programming on GPU
- Unit tests, maintainability and portability
- Performances of the code
- Multiple GPUs

Ongoing and future work

- Programming on GPU
- Unit tests, maintainability and portability
- Performances of the code
- Multiple GPUs

Ongoing and future work

- Programming on GPU
- Unit tests, maintainability and portability
- Performances of the code
- Multiple GPUs
The Hardware package

- Structure of the code
- Physics
- Hardware
- Programming on GPU
- Unit tests, maintainability and portability
- Performances of the code
- Multiple GPUs
- Ongoing and future work

Hardware package

- Code
- Open CL Buffers
- Hardware
- System
- Device
- OPEN CL Compiler
- TESTS

CL2QCD

---

Alessandro Sciarra (Goethe Universität)
The Hardware package

Lattice Seminar
DESY
Alessandro Sciarra

CL$^2$QCD features
Structure of the code
Physics
Hardware
Programming on GPU
Unit tests, maintainability and portability
Performances of the code
Multiple GPUs
Ongoing and future work

---

HARDWARE

CODE

\[ \psi\text{-fields} \]
\[ H_\mu\text{-fields} \]
\[ U_\mu\text{-fields} \]
\[ \text{Correlator} \]

\underline{HARDWARE}

\underline{BUFFERS}

\[ \psi\text{-fields} \]
\[ U_\mu\text{-fields} \]
\[ H_\mu\text{-fields} \]

\underline{OPEN CL}

\underline{BUFFERS}

\[ \text{PRNG} \]

\[ \text{Molecular Dynamics} \]

\[ \text{Heatbath} \]

\[ \text{Buffers} \]

\[ \text{Real} \]

\[ \text{Complex} \]

\underline{OPEN CL}

\underline{COMPILER}

\underline{DEVICE}

\underline{SYSTEM}

\underline{TESTS}

Alessandro Sciarra (Goethe Universität)
The Hardware package

Hardware

CODE

Complex

Hμ-fields

Uμ-fields

Correlator

PRNG

ψ-fields

3 × 3 matrix

8-vector

12-vector

3-vector

Plain

Hμ-fields

Uμ-fields

ψ-fields

Buffers

Heatbath

Molecular Dynamics

Kappa

PRNG

Buffers

OPEN CL BUFFERS

LATTICE BUFFERS

OPEN CL COMPILER

SYSTEM

DEVICE

TESTS

The Hardware package
Outline of the talk

1. Motivation and our philosophy

2. CL\textsuperscript{2}QCD features

3. Structure of the code
   - Physics
   - Hardware

4. Programming on GPU

5. Unit tests, maintainability and portability

6. Performances of the code
   - Multiple GPUs

7. Ongoing and future work
General Purpose Graphics Processing Unit

- CL$^2$QCD features
- Structure of the code
- Physics
- Hardware

Programming on GPU
- Unit tests, maintainability and portability
- Performances of the code
- Multiple GPUs
- Ongoing and future work
Lattice Seminar
DESY

ALESSANDRO SCIARRA

CL²QCD features
Structure of the code
Physics
Hardware

Programming on GPU
Unit tests, maintainability and portability
Performances of the code
Multiple GPUs
Ongoing and future work

General Purpose Graphics Processing Unit

GPU
### General Purpose Graphics Processing Unit

<table>
<thead>
<tr>
<th>CARD</th>
<th>CHIP</th>
<th>MEMORY (GB)</th>
<th>PEAK DP (GFLOPS)</th>
<th>PEAK BW (GB/s)</th>
<th>CLOCK (MHz)</th>
<th>YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD Radeon HD 5870</td>
<td>Cypress</td>
<td>1</td>
<td>544</td>
<td>154</td>
<td>850</td>
<td>2009</td>
</tr>
<tr>
<td>AMD Radeon HD 7970</td>
<td>Tahiti</td>
<td>3</td>
<td>947</td>
<td>264</td>
<td>925</td>
<td>2012</td>
</tr>
<tr>
<td>AMD FirePro S10000</td>
<td>Tahiti Pro GL</td>
<td>2 × 3 - 6</td>
<td>2 × 1480</td>
<td>2 × 240</td>
<td>825</td>
<td>2012</td>
</tr>
<tr>
<td>AMD FirePro S9050</td>
<td>Thaiti</td>
<td>12</td>
<td>806</td>
<td>264</td>
<td>900</td>
<td>2014</td>
</tr>
<tr>
<td>AMD FirePro S9150</td>
<td>Hawaii</td>
<td>16</td>
<td>2530</td>
<td>320</td>
<td>900</td>
<td>2014</td>
</tr>
<tr>
<td>AMD FirePro S9170</td>
<td>Grenada</td>
<td>32</td>
<td>2620</td>
<td>320</td>
<td>930</td>
<td>2015</td>
</tr>
<tr>
<td>AMD FirePro S9300</td>
<td>Capsaicin</td>
<td>2 × 4 (HBM)</td>
<td>868</td>
<td>2 × 512</td>
<td>850</td>
<td>2016</td>
</tr>
<tr>
<td>NVIDIA GeForce GTX 680</td>
<td>Kepler</td>
<td>2 - 4</td>
<td>129</td>
<td>192</td>
<td>1006</td>
<td>2012</td>
</tr>
<tr>
<td>NVIDIA Tesla K40</td>
<td>Kepler</td>
<td>12</td>
<td>1680</td>
<td>288</td>
<td>745</td>
<td>2013</td>
</tr>
<tr>
<td>NVIDIA Tesla K80</td>
<td>Kepler</td>
<td>2 × 12</td>
<td>2912</td>
<td>2 × 240</td>
<td>560</td>
<td>2014</td>
</tr>
</tbody>
</table>
General Purpose Graphics Processing Unit

<table>
<thead>
<tr>
<th>CARD</th>
<th>CHIP</th>
<th>MEMORY (GB)</th>
<th>PEAK DP (GFLOPS)</th>
<th>PEAK BW (GB/s)</th>
<th>CLOCK (MHz)</th>
<th>YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD Radeon HD 5870</td>
<td>Cypress</td>
<td>1</td>
<td>544</td>
<td>154</td>
<td>850</td>
<td>2009</td>
</tr>
<tr>
<td>AMD Radeon HD 7970</td>
<td>Tahiti</td>
<td>3</td>
<td>947</td>
<td>264</td>
<td>925</td>
<td>2012</td>
</tr>
<tr>
<td>AMD FirePro S10000</td>
<td>Tahiti Pro GL</td>
<td>2×3 - 6</td>
<td>2×1480</td>
<td>2×240</td>
<td>825</td>
<td>2012</td>
</tr>
<tr>
<td>AMD FirePro S9050</td>
<td>Thaïti</td>
<td>12</td>
<td>806</td>
<td>264</td>
<td>900</td>
<td>2014</td>
</tr>
<tr>
<td>AMD FirePro S9150</td>
<td>Hawaii</td>
<td>16</td>
<td>2530</td>
<td>320</td>
<td>900</td>
<td>2014</td>
</tr>
<tr>
<td>AMD FirePro S9170</td>
<td>Grenada</td>
<td>32</td>
<td>2620</td>
<td>320</td>
<td>930</td>
<td>2015</td>
</tr>
<tr>
<td>AMD FirePro S9300</td>
<td>Capsaicin</td>
<td>2×4 (HBM)</td>
<td>868</td>
<td>2×512</td>
<td>850</td>
<td>2016</td>
</tr>
<tr>
<td>NVIDIA GeForce GTX 680</td>
<td>Kepler</td>
<td>2 - 4</td>
<td>129</td>
<td>192</td>
<td>1006</td>
<td>2012</td>
</tr>
<tr>
<td>NVIDIA Tesla K40</td>
<td>Kepler</td>
<td>12</td>
<td>1680</td>
<td>288</td>
<td>745</td>
<td>2013</td>
</tr>
<tr>
<td>NVIDIA Tesla K80</td>
<td>Kepler</td>
<td>2×12</td>
<td>2912</td>
<td>2×240</td>
<td>560</td>
<td>2014</td>
</tr>
</tbody>
</table>

\[ \rho = \frac{\text{Number of FLOPs}}{\text{Number of Bytes to read and write}} \]

Wilson fermions \( \rho(\bar{\psi}) \sim 0.57 \)
Staggered fermions \( \rho(D_{KS}) \sim 0.35 \)

"FLOPS do not count." – Clark
CPU vs GPU code

\[ \phi = \alpha \psi_1 + \psi_2 \]
CPU vs GPU code

```
void saxpy(
    const spinor *x,
    const spinor *y,
    const hmc_complex *alpha,
    spinor *out)
{
    for(unsigned int index = 0; index < VTOT ; index += 1 )
    {
        const spinor tmp = spinor_times_complex(x[index], alpha);
        out[index] = spinor_acc(y[index], tmp);
    }
}
```

$\phi = \alpha \psi_1 + \psi_2$
CPU vs GPU code

\[
\phi = \alpha \psi_1 + \psi_2
\]

Alessandro Sciarra (Goethe Universität)

---

__kernel void saxpy(__global const spinor *x, __global const spinor *y, __global const hmc_complex *alpha, __global spinor *out) {
    int id = get_global_id(0);
    int gs = get_global_size(0);

    for(unsigned int idMem = id; idMem < SPINORFIELDSIZE_MEM; idMem += gs) {
        const spinor tmp = spinor_times_complex(x[idMem], alpha);
        out[idMem] = spinor_acc(y[idMem], tmp);
    }
}
Outline of the talk

1. Motivation and our philosophy

2. CL$^2$QCD features

3. Structure of the code
   - Physics
   - Hardware

4. Programming on GPU

5. Unit tests, maintainability and portability

6. Performances of the code
   - Multiple GPUs

7. Ongoing and future work
Crucial concepts

Robert C. Martin (2009), «Clean Code»

Kent Beck (2002), «Test Driven Development: By Example»
Crucial concepts

- Test each single part of code on its own
- **Unit tests** implemented using **BOOST** and **CMake** unit test frameworks
- **Regression tests** for the Open CL parts are **absolutely mandatory**
- LQCD functions are local → analytic results to test against can be calculated
- Avoid dependence of the tests on specific environments

Robert C. Martin (2009), «*Clean Code*»
Kent Beck (2002), «*Test Driven Development: By Example*»

---

Robert C. Martin (2009), «*Clean Code*»
Kent Beck (2002), «*Test Driven Development: By Example*»

- Test each single part of code on its own
- **Unit tests** implemented using **BOOST** and **CMake** unit test frameworks
- **Regression tests** for the Open CL parts are **absolutely mandatory**
- LQCD functions are local → analytic results to test against can be calculated
- Avoid dependence of the tests on specific environments
Crucial concepts

- Test each single part of code on its own
- **Unit tests** implemented using **BOOST** and **CMake** unit test frameworks
- **Regression tests** for the Open CL parts are **absolutely mandatory**
- LQCD functions are local $\rightarrow$ analytic results to test against can be calculated
- Avoid dependence of the tests on specific environments

Robert C. Martin (2009), «Clean Code»
Kent Beck (2002), «Test Driven Development: By Example»

![Pie chart showing developments and maintenance](chart.png)

Developing 33%
Refactoring 25%
Testing 33%
Optimisation 8%

More Development 2011-2013
More Refactoring 2014-2018

PRESENT for 2017

Crucial concepts

Robert C. Martin (2009), «Clean Code»
Kent Beck (2002), «Test Driven Development: By Example»

- Test each single part of code on its own
- **Unit tests** implemented using **BOOST** and **CMake** unit test frameworks
- **Regression tests** for the Open CL parts are **absolutely mandatory**
- LQCD functions are local $\rightarrow$ analytic results to test against can be calculated
- Avoid dependence of the tests on specific environments

![Pie chart showing developments and maintenance](chart.png)

Developing 33%
Refactoring 25%
Testing 33%
Optimisation 8%

More Development 2011-2013
More Refactoring 2014-2018

PRESENT for 2017
1 Motivation and our philosophy

2 CL²QCD features

3 Structure of the code
   - Physics
   - Hardware

4 Programming on GPU

5 Unit tests, maintainability and portability

6 Performances of the code
   - Multiple GPUs

7 Ongoing and future work

Outline of the talk
The Dirac operator kernel

Performance of Wilson $D$

Performance of Staggered $D_{KS}$

Lattice Size

GB/s

GFLOPS

Performance of Wilson $D$

Performance of Staggered $D_{KS}$

Lattice Size

GB/s

GFLOPS
**CL²QCD VS tmLQCD (in 2013)**

- Runs done on LOEWE-CSC
- tmLQCD has been run on a whole node (24 cores)
- Price per flop for the GPUs much lower than for the CPUs

M. Bach et al. «Twisted-Mass Lattice QCD using OpenCL»
http://pos.sissa.it/archive/conferences/187/032/LATTICE%202013_032.pdf

<table>
<thead>
<tr>
<th>SETUP</th>
<th>(m_\pi) (MeV)</th>
<th>(\beta)</th>
<th>(\kappa_c)</th>
<th>(N_\tau)</th>
<th>(N_\sigma)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>260</td>
<td>3.9</td>
<td>0.160856</td>
<td>8</td>
<td>24</td>
</tr>
<tr>
<td>B</td>
<td>310</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>520</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**AT MAXIMAL TWIST**

1 GPU ≈ 4 CPU
Outline of the talk

1. Motivation and our philosophy

2. CL$^2$QCD features

3. Structure of the code
   - Physics
   - Hardware

4. Programming on GPU

5. Unit tests, maintainability and portability

6. Performances of the code
   - Multiple GPUs

7. Ongoing and future work
CL$^2$QCD can use multiple GPUs within the same node

At the moment, the lattice can be divided only in the temporal direction

Tested on SANAM (AMD FirePro S10000, in 2013), 4 GPUs per node

**Multi-GPU scaling**

Hard scaling:
The total lattice size is kept constant

Weak scaling:
The local lattice size is kept constant
Outline of the talk

1. Motivation and our philosophy

2. $CL^2QCD$ features

3. Structure of the code
   - Physics
   - Hardware

4. Programming on GPU

5. Unit tests, maintainability and portability

6. Performances of the code
   - Multiple GPUs

7. Ongoing and future work
**Plausible future work**

- Pure Wilson RHMC
- Staggered pion mass
- Clover term
- Multiple pseudofermions
- Physics analytic tests
- Smearing?
- Disentangle all packages + work on CMake
- Parallelization in >1 directions

**Ongoing and future work**

- Arbitrary parallelization direction
- Multiple GPUs
- Ongoing and future work

**https://github.com/CL2QCD/cl2qcd**

---

**Lattice Seminar DESY**

**Alessandro Sciarra**

**CL²QCD features**

- Structure of the code
- Physics
- Hardware
- Programming on GPU
- Unit tests, maintainability and portability
- Performances of the code
- Multiple GPUs

**HIC for FAIR – 20.06.2016**

Alessandro Sciarra (Goethe Universität)
Plausible future work

https://github.com/CL2QCD/cl2qcd
Plausible future work

- Parallelization in >1 directions
- Disentangle all packages + work on CMake
- Multiple pseudofermions
- Pure Wilson RHMC
- Arbitrary parallelization direction
- Clover term
- Staggered pion mass
- Smearing?

https://github.com/CL2QCD/cl2qcd
Plausible future work

2016

- Arbitrary parallelization direction
- Pure Wilson RHMC
- Staggered pion mass

2017

- Multiple pseudofermions
- Clover term
- Smearing?

2018

- Disentangle all packages + work on CMake
- Parallelization in >1 directions

https://github.com/CL2QCD/cl2qcd
Plausible future work

https://github.com/CL2QCD/cl2qcd