# Portable LQCD Monte Carlo code using OpenACC

Giorgio Silvi

JSC - Forschungszentrum Juelich

**based on arXiv:1701.00426**

*C. Bonati, E. Calore, S. Coscetti, M. D'Elia, M. Mesiti, F. Negro, S. F. Schifano, G. S., R. Tripiccione*

Lattice 2017, Granada

The present panorama of HPC architectures make use mainly of:

**Conventional CPUs**

- Tens of fat cores
- Moderate vector units (2 to 4 elements)
- Hundreds of GigaFlops per processor

**Many-core CPUs**

- Several tens of slim cores
- Wide vector units (4 to 8 elements)
- Performance order of TeraFlops

**GPUs**

- Thousands of slim cores
- Extremely wide vectors (16 to 32 elements)
- Performance order of several TeraFlops

The present panorama of HPC architectures make use mainly of:

**Conventional CPUs**

- Tens of fat cores
- Moderate vector units (2 to 4 elements)
- Hundreds of GigaFlops per processor

**Many-core CPUs**

- Several tens of slim cores
- Wide vector units (4 to 8 elements)
- Performance order of TeraFlops

**GPUs**

- Thousands of slim cores
- Extremely wide vectors (16 to 32 elements)
- Performance order of several TeraFlops

**Computing centers today have NOT reached a common consensus on the "best" processor option.**

## Exploit parallelism and portability

In this scenario, the development of applications would greatly benefit from a unique version able to offer portability:

## Exploit parallelism and portability

In this scenario, the development of applications would greatly benefit from a unique version able to offer portability:

We develope a full state-of-the-art code for Lattice QCD simulations with **tree-level improved gauge action** and **stout-smeared action for staggered fermions**, using:

- OpenACC directive-based programming model
- PGI compiler (16.10)

And test performances on different processor architectures

## Why OpenACC?

- Directive based (like OpenMP)
- Wide compiler support
- Hardware agnostic (targeting mainly GPUs though)
- Overhead to offload code limited to few pragma lines
- Code can still be compiled as plain C, ignoring *pragma* lines

## Numerical algorithms for LQCD: brief overview

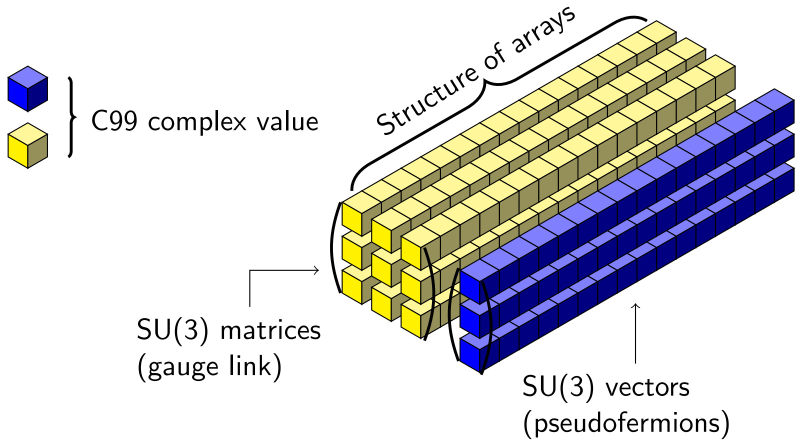
We aim to evaluate integrals of the form:

$$\langle O \rangle = \int \mathscr{D}U \mathscr{D}\phi \mathscr{D}H \, O[U] \exp\left(-\frac{1}{2}H^2 - S_g[U] - \phi^* M[U]^{-1/4}\phi\right)$$

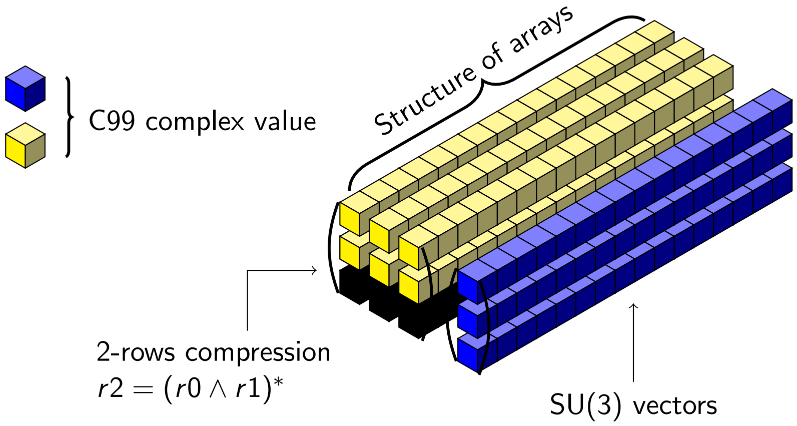
The fundamental data that need to be stored are:

- $U$: **gauge fields**
- $\phi$: **pseudofermion fields**

## Data Structure: SoA



## Data Structure: SoA



### Lexicographical ordering

$$\mathrm{idxh} = \ (\mathsf{int})\frac{x_0 + \mathsf{LNH\_N0}[x_1 + \mathsf{LNH\_N1}(x_2 + \mathsf{LNH\_N2}\, x_3)]}{2} \tag{1}$$

we allow for full freedom in the mapping of the physical directions $x, y, z$ and $t$ onto the logical directions $x_0, x_1, x_2$ and $x_3$

## Details of kernels

```
for(d3=0; d3<nd3;d3++) {

  for(d2=0; d2<nd2; d2++) {
    for(d1=0; d1<nd1; d1++) {
     for(hd0=0; hd0 < nd0h; hd0++) {
          ...
      }
    }
  }
}
```

4-dimension nested loops

## Details of kernels

```
#pragma acc kernels present(in) present(out)present(u)
   present(backfield) async(1)

  for(d3=0; d3<nd3;d3++) {

    for(d2=0; d2<nd2; d2++) {
      for(d1=0; d1<nd1; d1++) {
       for(hd0=0; hd0 < nd0h; hd0++) {
          ...
        }
      }
    }
  }
```

**present()** - identify the data structures already present in the accelerator memory

## Details of kernels

```
#pragma acc kernels present(in) present(out)present(u)
   present(backfield) async(1)
 #pragma acc loop independent gang(GANG)
for(d3=0; d3<nd3;d3++) {

  for(d2=0; d2<nd2; d2++) {
    for(d1=0; d1<nd1; d1++) {
     for(hd0=0; hd0 < nd0h; hd0++) {
        ...
      }
    }
  }
}
```

**independent** - make the compiler aware of the data independence of loops iterations

## Details of kernels

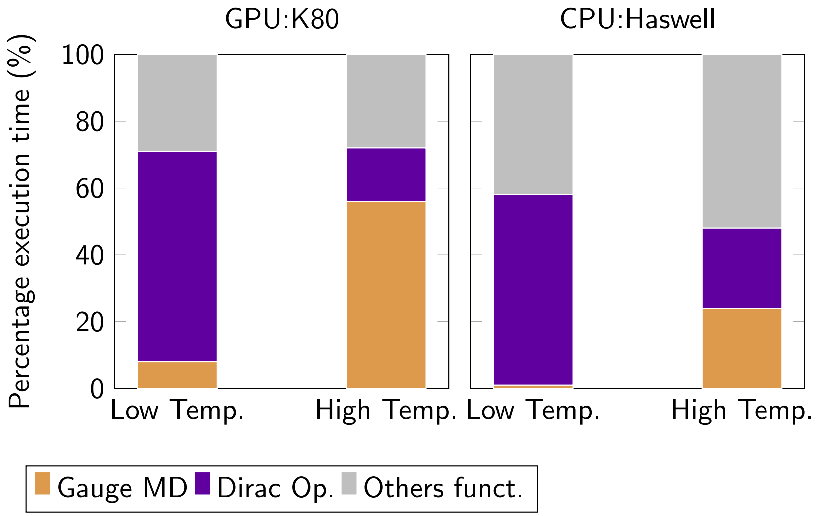
```
  #pragma acc kernels present(in) present(out)present(u)
     present(backfield) async(1)
  #pragma acc loop independent gang(GANG)
 for(d3=0; d3<nd3;d3++) {
    # pragma acc loop independent vector
       tile(TILE0,TILE1,TILE2)
   for(d2=0; d2<nd2; d2++) {
     for(d1=0; d1<nd1; d1++) {
      for(hd0=0; hd0 < nd0h; hd0++) {
         ...
       }
     }
   }
 }
```

**tile** - request to group iterations in order to execute them in the same (or close) compute units

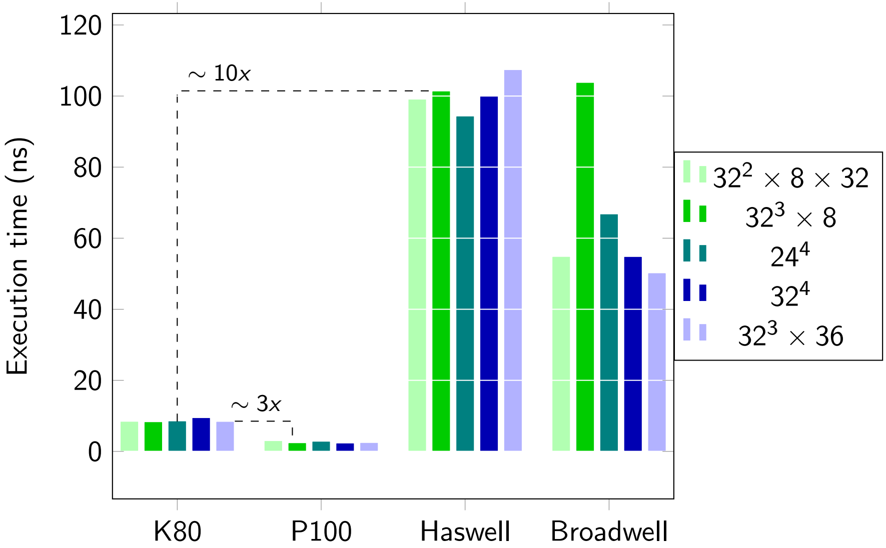
## Hardware

| | Xeon E5-2630 v3 | Xeon E5-2697 v4 | K80-GK210 | P100 |
|---|---|---|---|---|
| Year | 2014 | 2016 | 2014 | 2016 |
| Architetcure | Haswell | Broadwell | Kepler | Pascal |
| #physical-cores / SMs | 8 | 18 | 13 $\times$ 2 | 56 |
| #logical-cores / CUDA-cores | 16 | 36 | 2496 $\times$ 2 | 3584 |
| Nominal Clock (GHz) | 2.4 | 2.3 | 562 | 1328 |
| Nominal DP performance (Gflops) | $\approx$ 300 | $\approx$ 650 | 935 $\times$ 2 | 4759 |
| LL cache (MB) | 20 | 45 | 1.68 | 4 |
| Total memory supported (GB) | 768 | 1540 | 12 $\times$ 2 | 16 |
| Peak mem. BW (ECC-off) (GB/s) | 69 | 76.8 | 240 $\times$ 2 | 732 |

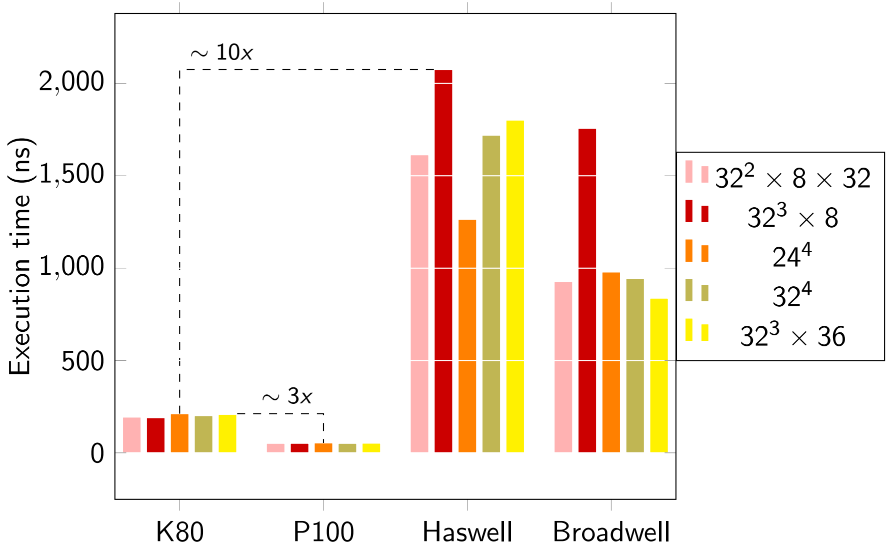
## Heavy steps of the code (at physical pion mass)



## Dirac Operator time per lattice site (DP)

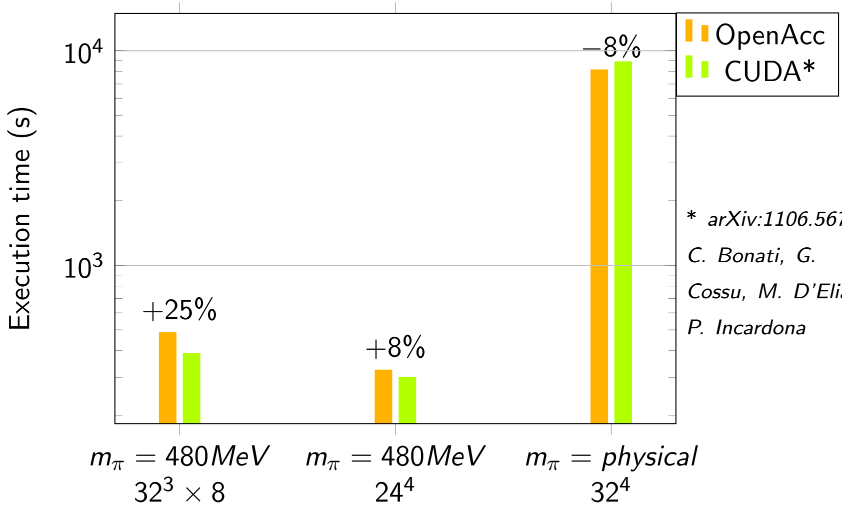


## Molecular Dynamics step time per lattice site



## OpenACC vs CUDA: full trajectory MC on K80

Wilson action + unimproved staggered fermions



* *arXiv:1106.5673 C. Bonati, G. Cossu, M. D'Elia, P. Incardona*

## Conclusions

**Remarks** :

- PGI compiler actual version (16.10) is not completely mature on KNL and AMD GPUs
- On both GPUs and CPUs we measured comparable levels of efficiency.
- Also performances are roughly comparable to an equivalent code written in CUDA
- the LQCD code is portable on a large subset of HPC relevant architectures.

**Future**:

- Assess performances on AMD and KNL systems
- Determine whether OpenMP4 provides same level of portability
- Already working on an MPI parallel version of our code able to run on large clusters of CPUs and accelerators.