Experiences in porting Lattice QCD kernels to GPUs

Giannis Koutsou
for
The CyQuarks Group
A. Abdel-Rehim, K. Hadjiyiannakou, A. Vaquero, J. Volmer
Mentors: Matt Norman and Hoshino Tetsuya

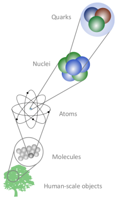
Computational-based Science and Technology Research Center (CaSToRC), The Cyprus Institute

Eurohack 2015, July 2015, Lugano, Switzerland

The Cyprus Institute

CaSToRC

Lattice QCD



QCD: Fundamental theory of strong interactions

- Part of the Standard Model of particle physics
- Responsible for binding quark into protons and neutrons
- Accounts for baryonic mass observed in the universe

- **Post-diction of experimentally well known quantities**
  - Masses of low-lying hadrons
  - Nucleon axial-charge g_A
  - Nucleon momentum fraction $\langle x \rangle$
  - Nucleon electromagnetic form-factors

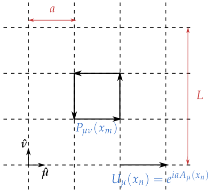
  "Benchmark" or "Gold-plated" quantities

- **Pre-diction of experimentally less (or not) known quantities**
  - Nucleon scalar and tensor charges g_S, g_T
  - Nucleon sigma terms
  - Axial charges of hyperons
  - Neutron electric dipole moment
  - Precision measurements of proton radius, anomalous magnetic moment of muon (hadronic)

  Contact to new physics

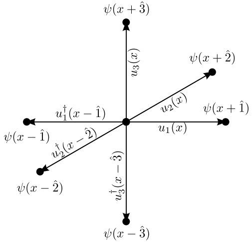
QCD on the lattice

4D space - time lattice



- 4D grid, spacing a, extent L
- Quark fields $\psi(x)$, $\bar{\psi}(x)$, gluon fields $U_\mu(x)$
- Finite $a \rightarrow$ UV cut-off
- Finite $L \rightarrow$ quantized momenta $\frac{2\pi}{L}\vec{n}$

- Monte Carlo simulation for ensembles of gluon fields, with probability $e^{S[U,M^{-1}]}$ (Euclidean time)
- Observables: $\langle O \rangle = \sum_{\{U\}} O(M^{-1}, U_\mu)$, with M^{-1} the quark propagator
- M, discrete covariant derivative (Dirac operator) $\rightarrow$ 4D stencil



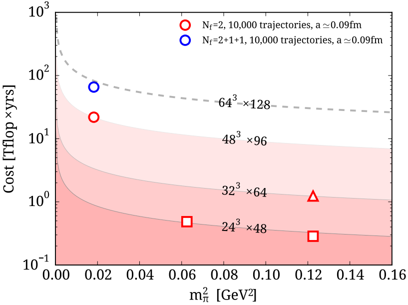
Simulation parameters

- Lattice spacing a: take limit $a \to 0$
- Quark mass m_q: take limit $m_q \to m_q^{\text{phys}}$
- Volume L: take limit $L \to \infty$

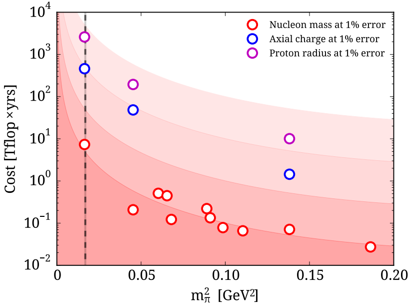
Empirical cost formula:

$$C^{\text{sim}} \propto \left(\frac{0.3\text{GeV}}{m_{\text{PS}}}\right)^{C_m}\left(\frac{L}{2\text{fm}}\right)^{C_L}\left(\frac{0.1\text{fm}}{a}\right)^{C_a}$$

Simulation cost



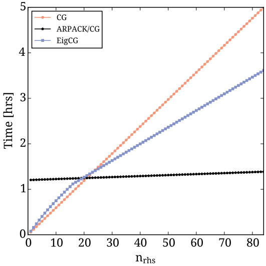
Analysis cost



Typical workflow

- **Pre-process**
  - — Prepare linear system right-hand-sides
  - — Custom code depending on problem/observable desired
  - → more maintainable and flexible code required: candidate for OpenACC
- **Production**
  - — Invert for right-hand-sides
  - — Conjugate-Gradient or variant employed
  - — Various preconditioning algorithms on the market
  - — Eigenvalue deflation typically offer $\sim 10\times$ speed-ups
  - → Use of optimized libraries in CUDA - QUDA package
- **Post-process/analysis**
  - — Generally site-local operations
  - — Multiplications between arrays of small matrices e.g. arrays of 12×12 complex matrices
  - → more maintainable and flexible code: candidate for OpenACC

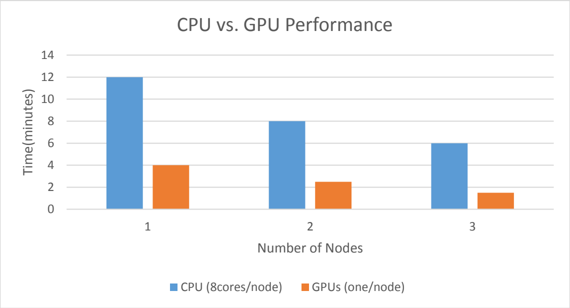
Matrix inversion on multiple right-hand-sides



- **Eigenvalue deflation**
  - $\mathcal{O}(1000)$ eigenvectors calculated for $10^7 \times 10^7$ matrix
  - CPU code employed up to now
- **Inversion**
  - Use of eigenvalues for preconditioning the inversion
  - QUDA used, eigenvectors read from disk

- **Goals during Eurohack: Perform deflation on GPUs**
  - Main challenge: limited GPU memory
  - Solution: Keep eigenvectors on host memory, stream in to GPU for matrix application

Timings



- Strong scaling of smaller-than-production problem size
- Scaling important: more GPUs means more memory meaning more eigenvectors

Analysis kernels

- **Many unrolled sparse-matrix times dense-matrix multiplications**
  - Write as loops to allow vectorisation
  - Generate index arrays of non-zero elements
- **Short, inline and static functions for matrix multiplications**
  - Need to rewrite as macros inline functions
- **Arrays of structs of arrays**
  - Care when transferring C-structures to GPU
- **Mainly using PGI compiler**
  - craycc issue when allocating device pointer

Analysis kernels

```c
/* multiply prop by \gamma_0 from the left */
static void
prop_g0_G(_Complex double out[NS*NC][NS*NC], _Complex double in[NS*NC][NS*NC])
{
  out[ 0][ 0] = -in[ 6][ 0];
  out[ 0][ 3] = -in[ 6][ 3];
  out[ 0][ 6] = -in[ 6][ 6];
  out[ 0][ 9] = -in[ 6][ 9];
  out[ 3][ 0] = -in[ 9][ 0];
  out[ 3][ 3] = -in[ 9][ 3];
  out[ 3][ 6] = -in[ 9][ 6];
  out[ 3][ 9] = -in[ 9][ 9];
  out[ 6][ 0] = -in[ 0][ 0];
  out[ 6][ 3] = -in[ 0][ 3];
  out[ 6][ 6] = -in[ 0][ 6];
```

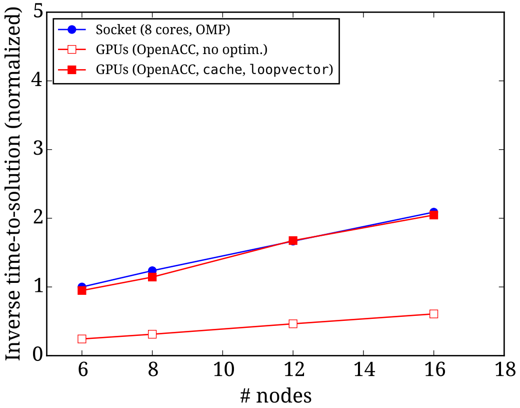
```c
/* multiply prop by \gamma_0 from the left */
static inline void
prop_g0_G(_Complex double out[NS*NC][NS*NC], _Complex double in[NS*NC][NS*NC])
{
  for(int i=0; i<NC*NS; i++)
     for(int j=0; j<NC*NS; j++) {
        out[i][j] = 0.;
        for(int k=0; k<prop_g0_G_nva; k++)
          out[i][j] += (prop_g0_G_val[i][j][k])*in[prop_g0_G_idx[i][j][k][0]][prop_g0_G_idx[i][j][k][1]];
     }

  return;
}

/* multiply prop by ^Har{\gamma_0} from the right */
```

```c
/* multiply prop by \gamma_0 from the left */
#define prop_g0_G_(out, in)                                         \
  for(int i=0; i<NC*NS; i++)                                        \
    for(int j=0; j<NC*NS; j++) {                                    \
      out[i][j] = 0.;                                               \
      for(int k=0; k<prop_g0_G_nva; k++)                            \
        out[i][j] += (prop_g0_G_val[i][j][k])*in[prop_g0_G_idx[i][j][k][0]][prop_g0_G_idx[i][j][k][1]]; \
    }                                                               \

/* multiply prop by \bar{\gamma_0} from the right */
#define prop_G_g0_(out, in)                                         \
  for(int i=0; i<NC*NS; i++)                                        \
    for(int j=0; j<NC*NS; j++) {                                    \
```

Analysis kernels

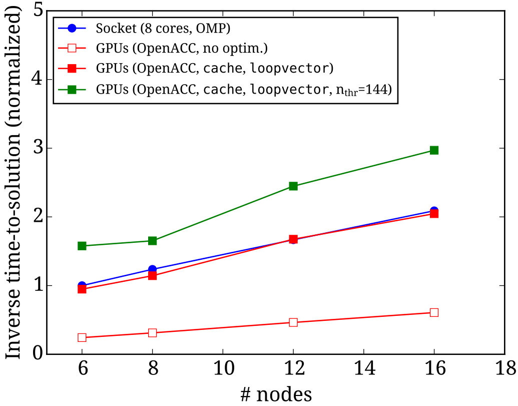
```
#endif
#ifdef QHG_ACC
#pragma acc enter data pcopyin(fwd[0:NC*NS], bwd[0:NC*NS]) async(1)
  for(int cs=0; cs<NC*NS; cs++) {
#pragma acc enter data pcreate(fwd[cs].field[0:lvol_bvol*NC*NS]) async(1)
#pragma acc update device(fwd[cs].field[0:lvol_bvol*NC*NS]) async(1)

#pragma acc enter data pcreate(bwd[cs].field[0:lvol_bvol*NC*NS]) async(1)
#pragma acc update device(bwd[cs].field[0:lvol_bvol*NC*NS]) async(1)
  }

#pragma acc enter data pcreate(corrC[0:lvol*SITE_SIZE]) async(1)

#pragma acc parallel loop gang vector_length(144)       \
  private(F, B, T, gF, fwd_tbc, bwd_tbc, tsrc)          \
  present(fwd, bwd, corrC) async(1)
#endif
  for(int v=0; v<lvol; v++) {
#pragma acc cache(                                \
                  F[0:(NC*NS)][0:(NC*NS)],        \
                  B[0:(NC*NS)][0:(NC*NS)],        \
                  gF[0:(NC*NS)][0:(NC*NS)],       \
                  T[0:(NC*NS)][0:(NC*NS)])

#pragma acc loop vector collapse(2)
    prop_load_(F, fwd, v);
#pragma acc loop vector collapse(2)
    prop_load_(B, bwd, v);

    int t = v/lv3;
    int gt = t + t0;
    if(gt < tsrc) {
#pragma acc loop vector collapse(2)
      prop_scale_(fwd_tbc, F);
#pragma acc loop vector collapse(2)
      prop_scale_(bwd_tbc, B);
    }
```

Timings



- Including any explicit `vector_length()` leads to wrong results
- Some speed-up may be possible, e.g. `vector_length(64)` produced a faster kernel

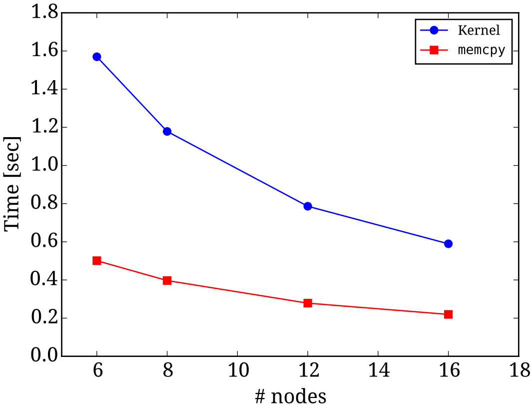
Timings



— Including explicit `vector_length()` leads to wrong results

— Some speed-up may be possible, e.g. various choices of `vector_length()` produced a faster kernel, but results are not always identical

Timings



- ~25% time in memcpy
- Production kernels will be larger ⇒ potential for speed-up
- Buggy reduction over threads

Impressions

- **OpenACC impressions**
  - Changes to code required (some unpleasant)
    - Un-unroll manually unrolled loops
    - Introduction of index arrays
    - Change inline functions to macros
    - → No impact on CPU code
  - Maintainability
    - Can switch between OpenACC/OpenMP with `ifdefs`
- **Hackathon impressions**
  - Excellent opportunity to concentrate on code development
  - Much more useful than schools which typically include only short example codes
  - Mentors at table were critical to go forward
    - Perhaps an extra day to ease-in would be helpful
    - Or as suggested to have a communication to start some OpenACC development a couple of weeks before Hackathon
    - Open to general GPU development (CUDA?)

Thank you!

A. Abdel-Rehim, C. Alexandrou, K. Hadjiyiannakou, A. Vaquero, J. Volmer

EUROPEAN UNION REPUBLIC OF CYPRUS Research Promotion Foundation STRUCTURAL FUNDS of the European Union in Cyprus our ideas, actions for development

The Project GPU Clusterware (ΤΠΕ/ΠΛΗΡΟ/0311(BIE)/09)