QUICK QEX Q&A

JAMES C OSBORN

XIAO-YONG JIN

USQCD Software Meeting
BNL
April 27, 2019

Quantum Expressions (QEX)

(https://github.com/jcosborn/qex)

- Framework for Lattice Field Theory
- Using Nim language (https://nim-lang.org)
- Python-like syntax, powerful metaprogrammming
- Designed for easy-to-use, high-level, optimized expression-based development

Simple Example

```
import qex

qexInit()


var lat = [4,4,4,4]

var lo = newLayout(lat)

echoAll "Hello from rank ", lo.myRank,
  " of ", lo.nRanks


var v1 = lo.ColorVector()

var v2 = lo.ColorVector()

var m1 = lo.ColorMatrix()
```

```
threads:
  m1 := 1
  v1 := 2
  v2.even := m1 * v1
  v2.odd := 3
  echo "v2 even: ", v2.even.norm2
  echo "v2 odd: ", v2.odd.norm2


threads:
  shift(v1, dir=0, len=1, v2)
  shift(v2, dir=3, len=2, v1)
  echo "v2 even: ", v2.even.norm2
  echo "v2 odd: ", v2.odd.norm2


qexFinalize()
```

Another Example

```
import qex
proc plaq*(g: seq[Field]): float =
  let lo = g[0].l
  let nd = lo.nDim
  var m = lo.ColorMatrix()
  var tr: type(trace(m))
  let t = newTransporters(g, g[0], 1)
  threads:
    m := 0
    for mu in 1..<nd:
      for nu in 0..<mu:
        m += (t[mu]^*g[nu]) * (t[nu]^*g[mu]).adj
    tr = trace(m)
  result = tr.re/(0.5*float(nd*(nd-1)*lo.physVol))
```

```
qexInit()
var lat = intSeqParam("lat",
  @[4,4,4,4])
var lo = newLayout(lat)
var g = lo.newGauge()
threads:
  g.unit
echo "average plaq: ", plaq(g)
qexFinalize()
```

```
> mpirun -np 4 ./plaq
>   average plaq: 3.0
```

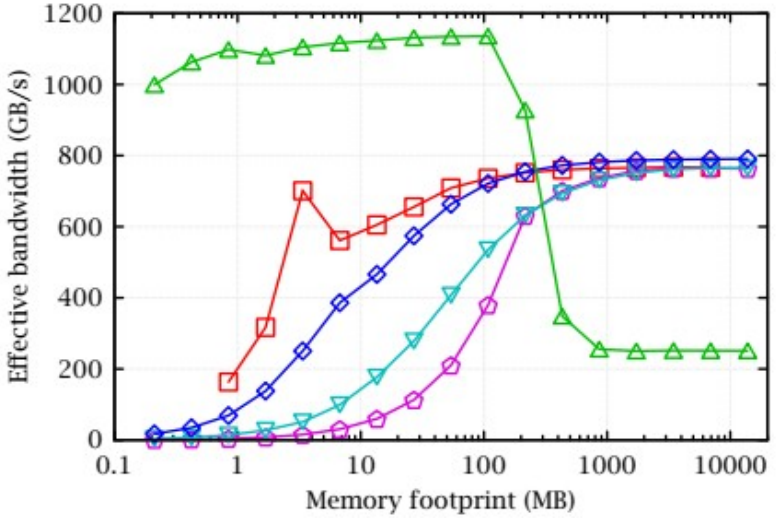
compile with “-d:Nc:9”

```
> mpirun -np 4 plaq -lat:8,8,8,8,8
>   average plaq: 9.0
```

Status

- Under active development, but basic framework fairly complete
- Optimized for BG/Q and KNL, also using QUDA for NVIDIA GPUs
- Currently being used in production for LSD 8f staggered analysis on Lassen
- Also used for HMC tests for ECP Critical Slowing Down (QNHMC)

GPU Development



OpenCL, $V = 32, M = 1$
OpenMP clang, $V = 32, M = 1$
OpenMP xlc, $V = 64, M = 1$
CUDA gcc, $V = 16, M = 2$
2 POWER9 gcc, $V = 4, M = 1$

```
proc test(V, M: static[int]; N: int) =
  var
    x = newColorMatrixArray(V,M,N)
    y = newColorMatrixArray(V,M,N)
    z = newColorMatrixArray(V,M,N)

  threads:
    x := 0
    y := 1
    z := 2
    timeit "CPU": x += y * z

  timeit "GPU5":
    onGpu(N, 32):
      x += y * z
  timeit "GPU6": onGpu(N, 64): x += y * z
  timeit "GPU7": onGpu(N, 128): x += y * z

  threads: timeit "CPU": x += y * z
```

Define complex 3×3 matrix field

CPU threads

Set diagonal elements

$x^l_{ij} \leftarrow x^l_{ij} + y^l_{ik} \times z^l_{kj}$

Benchmark on CPU

Run statement block on GPU

Rerun with different T/B

Back to CPU threads

- GPU support:
  working examples using CUDA, OpenCL, OpenMP
- Will continue to experiment with different backends and integrate into framework

Analysis Development

- Building pieces for perambulator-based analysis (current targeting Summit)
  - Read Laplacian eigenvalues on a timeslice
  - Run solver (e.g. QUDA MG)
  - Distribute solution to chopped lattice
  - Perform contractions (pack into BLAS calls)

  (done; in progress; to do)

Other Development Plans

- Have flexible MG working for Wilson;
  will experiment with tuning algorithm, extend to staggered
- Plug in more external routines (QUDA, Grid, …)
- Continue developing native (Nim) expression optimizations (have prototype for optimization of expressions with shifts)
- Interested in feedback on other potential use cases

- Questions?
- Suggestions?