

# QEX: a framework for lattice field theories

---

Xiao-Yong Jin & James C. Osborn

Argonne Leadership Computing Facility, Argonne National Laboratory



U.S. DEPARTMENT OF  
**ENERGY**

Office of  
Science

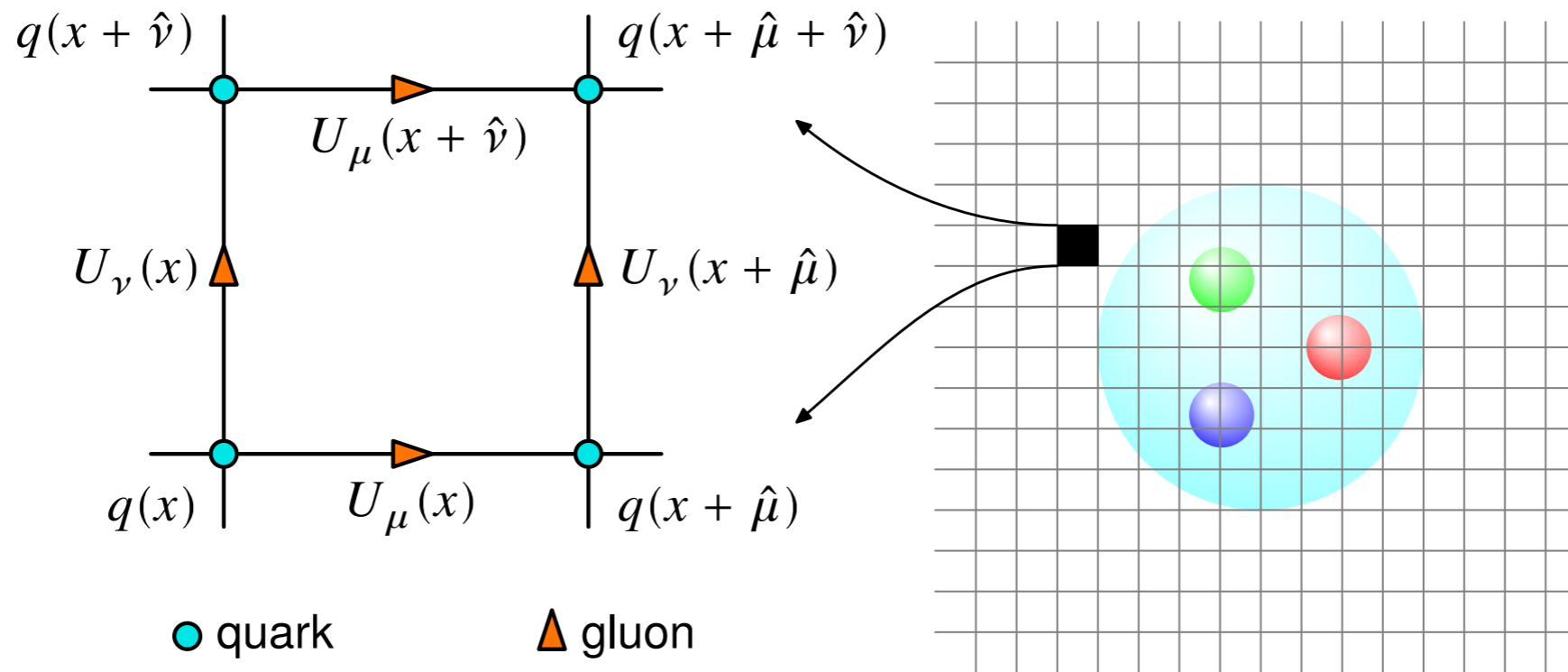
Argonne   
NATIONAL LABORATORY

# Outline

---

- Intro
- Nim
- QEX: Quantum EXpressions
- Benchmark
- Outro

# Lattice gauge theory



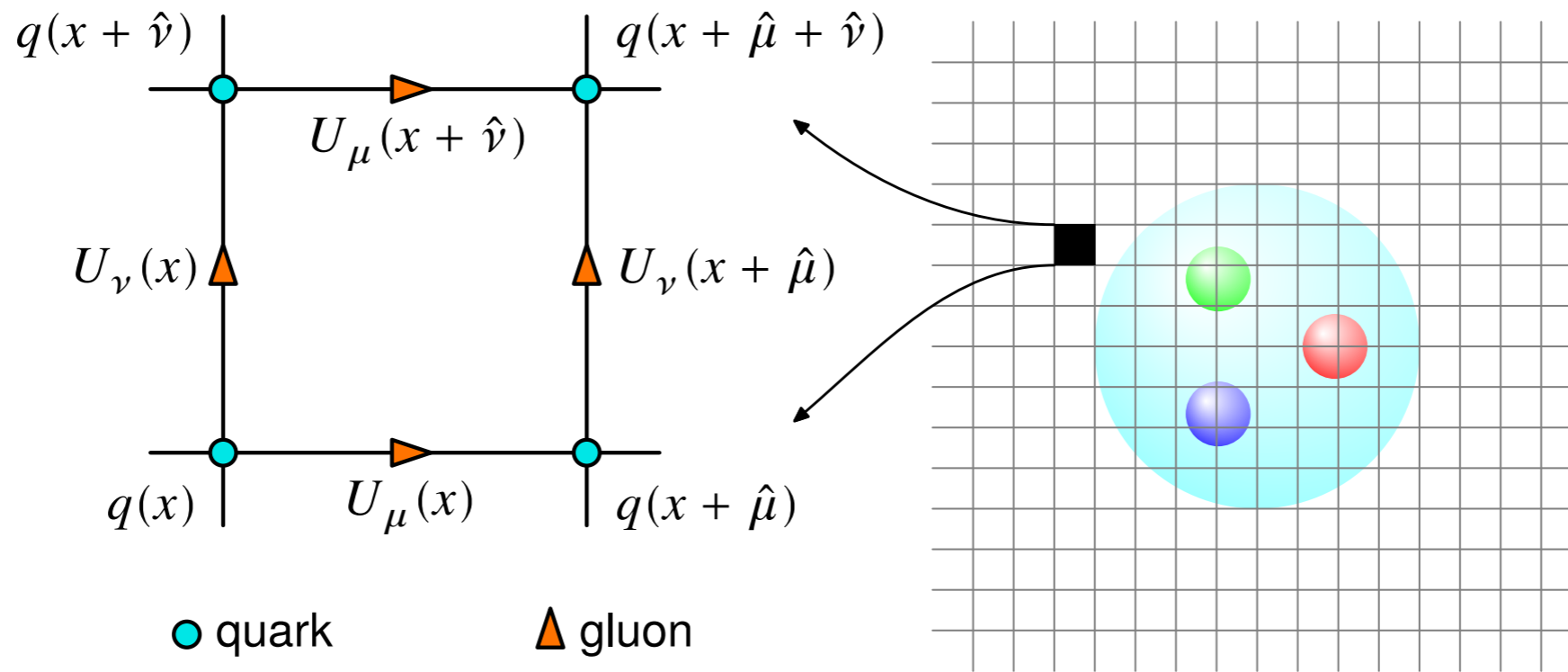
$$\langle O(U, q, \bar{q}) \rangle = \frac{1}{Z} \int [dU] \prod_f \det[M_f] O(U, M^{-1}) e^{-S_g}$$

$$\equiv \langle O(U, M^{-1}[U]) \rangle_U$$

$$\approx \frac{1}{N} \sum_{k=1}^N O(U^{(k)})$$

$$\text{Pr}(U) \propto e^{-S_g} \prod_f \det[M_f]$$

# Lattice gauge theory



$$\langle O(U, q, \bar{q}) \rangle = \frac{1}{Z} \int [dU] \prod_f \det[M_f] O(U, M^{-1}) e^{-S_g}$$

**Physical  
Observable**

$$\equiv \langle O(U, M^{-1}[U]) \rangle_U$$

$$\approx \frac{1}{N} \sum_{k=1}^N O(U^{(k)})$$

$$\text{Pr}(U) \propto e^{-S_g} \prod_f \det[M_f]$$

**Average Value of  
Configurations  
from  
Monte Carlo**

# Lattice gauge theory *NOT* for QCD only

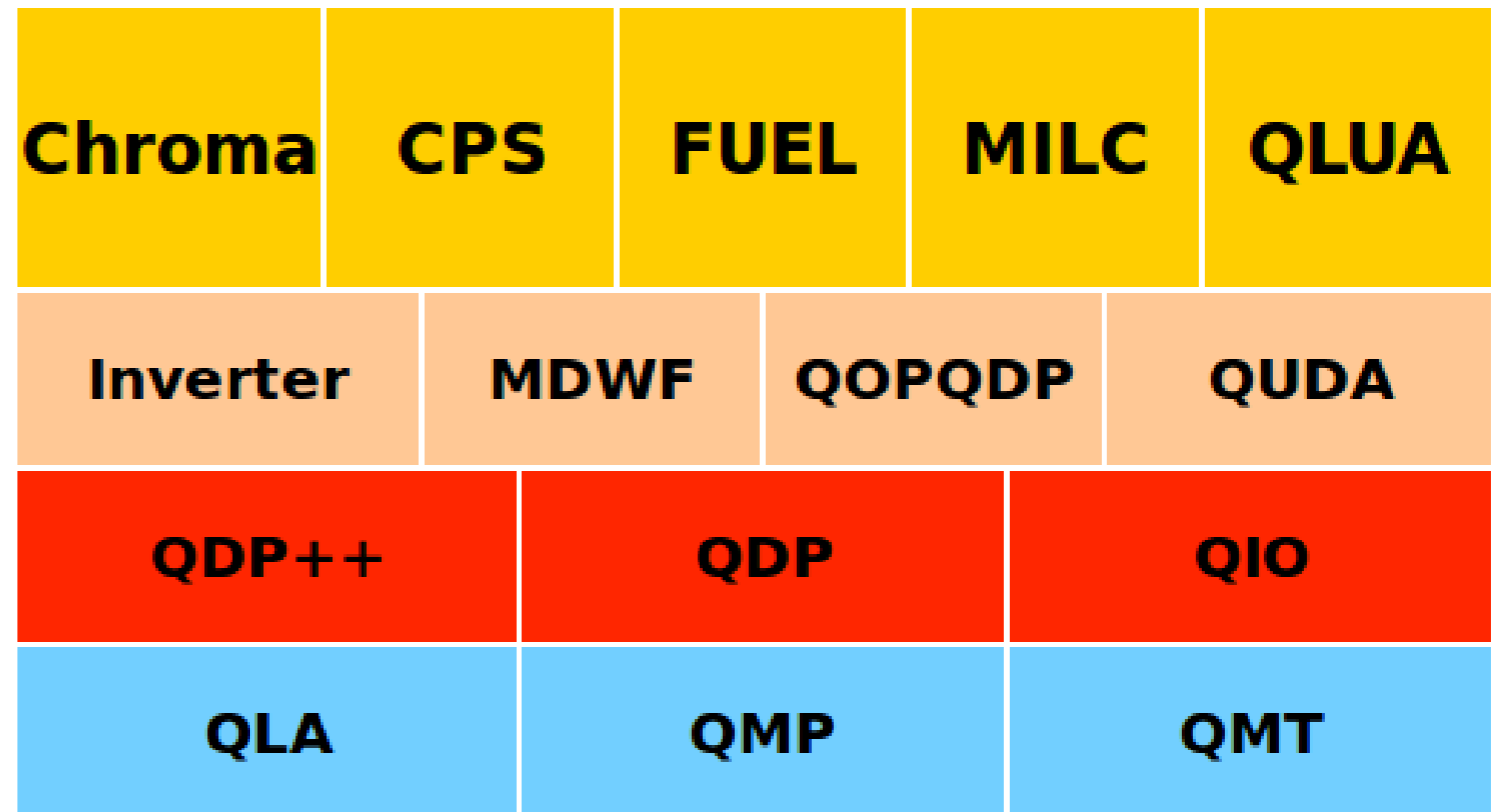
---

- The main target is standard model QCD
- In addition:
  - Various gauge groups, fermion representations
- BSM theoretical advancement:
  - Muon  $g-2$
  - Dark matter phenomenology
  - New strong coupled theories

# USQCD SciDAC software

---

- Mostly C/C++
- Some Perl generated code
- Some Lua (FUEL, QLUA)
  - Ease of use
  - Rapid development
  - Speed of C
- New framework QLL (J.C.Osborn)
  - Hand-written + Lua generated C
  - Well tuned staggered + Naik CG 23% peak on BG/Q



# HPC coding “dream”

---

- High level constructs & ease of low level manipulations
- Optimized code from natural expressions
- Compile time reflection
  - Controlled optimization using compile time information
- Type safety and type inference

# Nim



- Modern (since 2008) language
- “Efficient Expressive Elegant”
- Statically typed systems language (full access to low-level objects & code) with type inference
- Generates C or C++ code & compile with any compiler
- Integrated build system (no Makefile necessary): copy main program, modify, compile
- <http://nim-lang.org>

# Nim — both low-level and high-level

---

- Low-level efficiency
  - Can manually manage memory instead of GC
  - Cross module inlining and constant unfolding
  - Whole program dead code elimination
- High-level wrappers & libraries
  - gmp, bignum, nimblas, linalg(LAPACK), ...
  - bindings to GTK2, the Windows API, the POSIX API, OpenGL, SDL, Cairo, Python, Lua, TCL, X11, libzip, PCRE, libcurl, mySQL, SQLite, ...
- NimScript: shell-like scripting
  - Used in compiler for compile-time evaluation
  - Available to plug in to application and can interface with rest of application

# Nim – Generics & meta-programming

---

C++	Nim
Preprocessor macros	Templates: in-line code substitutions, also allows overloading, completely hygienic (if desired)
Templates	Generics: applies to types, procedures, templates, and macros also allows type-classes, concepts
N/A	Macros: similar to Lisp: syntax tree of arguments passed to macro at compile time

# Nim – Compile time reflection

---

```
import macros
proc f[T](x:T):T = x*x           # Works for any T that has `*`

macro m(x:typed):untyped =     # Run at compile time
  echo x.treeRepr              # → Call
                               #   Sym "f"
                               #   Float64Lit 2.0
  echo x[0].getTypeImpl.repr   # → proc (x: float64): float64
  echo x[0].symbol.getImpl.repr # → proc f(x: T): T =
                               #   result = x * x
  result = newCall("f",x)      # Set and print the result:
  echo result.repr             # → f(f(2.0'f64))

echo m(f(2.0))                 # Run time prints: 16.0
```

# New framework: QEX (Quantum EXpressions)

---

- Mostly in Nim
- Using QLL(layout & comm), QIO, QMP
  - Nim works great with C
- Working staggered solver and meson analysis
- High level interface in development
- Available on <https://github.com/jcosborn/qex>

# QEX – Tensor Programming Library in development

```
tensorOps:  
  v2 = 0  
  v2 += v1 + 0.1  
  v3 += m1 * v2
```

macro sees



```
StmtList  
  Asgn  
    Ident !"v2"  
    IntLit 0  
  Infix  
    Ident !"+="  
    Ident !"v2"  
    Infix  
      Ident !"+"  
      Ident !"v1"  
      Float64Lit 0.1  
  # ...
```



macro writes

```
for j in 0..2:  
  v2[j] = 0  
  v2[j] += v1[j] + 0.1  
  for k in 0..2:  
    v3[k] += m1[k,j] * v2[j]
```

# QEX—Low level optimization

---

```
var t:array[3,tuple[re:vec4double, im:vec4double]]
...
t[0].re = ...
t[0].im = ...
...
```



```
var t0re: vec4double
var t0im: vec4double
...
t0re = ...
t0im = ...
...
```

# QEX—Mapping intrinsics

---

```
proc mm512_mul_pd*(a: m512d; b: m512d): m512d
  {.importc: "_mm512_mul_pd", header: "immintrin.h".}
template basicDefs(T, F, N, P, S: untyped): untyped {.dirty.} =
  # ...
  template mul*(x, y: T): T = `P "_mul_" S` (x, y)
  # ...
basicDefs(m512d, float64, 8, mm512, pd)
```

# QEX—Handling OpenMP

---

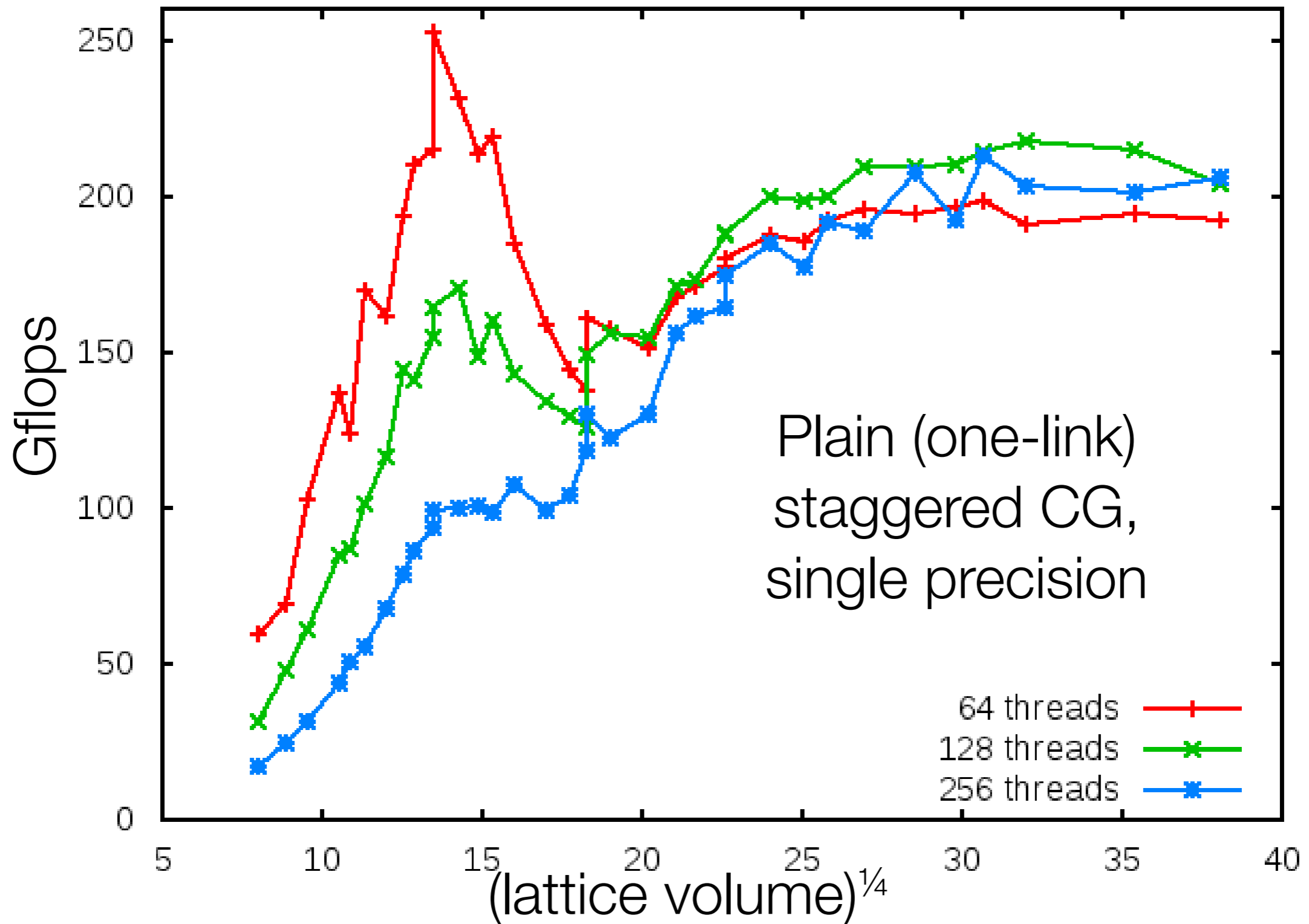
```
const ompFlag = "-fopenmp" # defined from build system
{. passC: ompFlag .}
{. passL: ompFlag .}
{. pragma: omp, header: "omp.h" .}
proc omp_set_num_threads*(x: cint) {.omp.}
proc omp_get_num_threads*(): cint {.omp.}
proc omp_get_thread_num*(): cint {.omp.}
template ompPragma(p:string):untyped =
  {. emit: "#pragma omp " & p .}
template ompBarrier* = ompPragma("barrier")
template ompBlock(p:string; body:untyped):untyped =
  ompPragma(p)
  block:
    body
```

# QEX—Performance on KNL

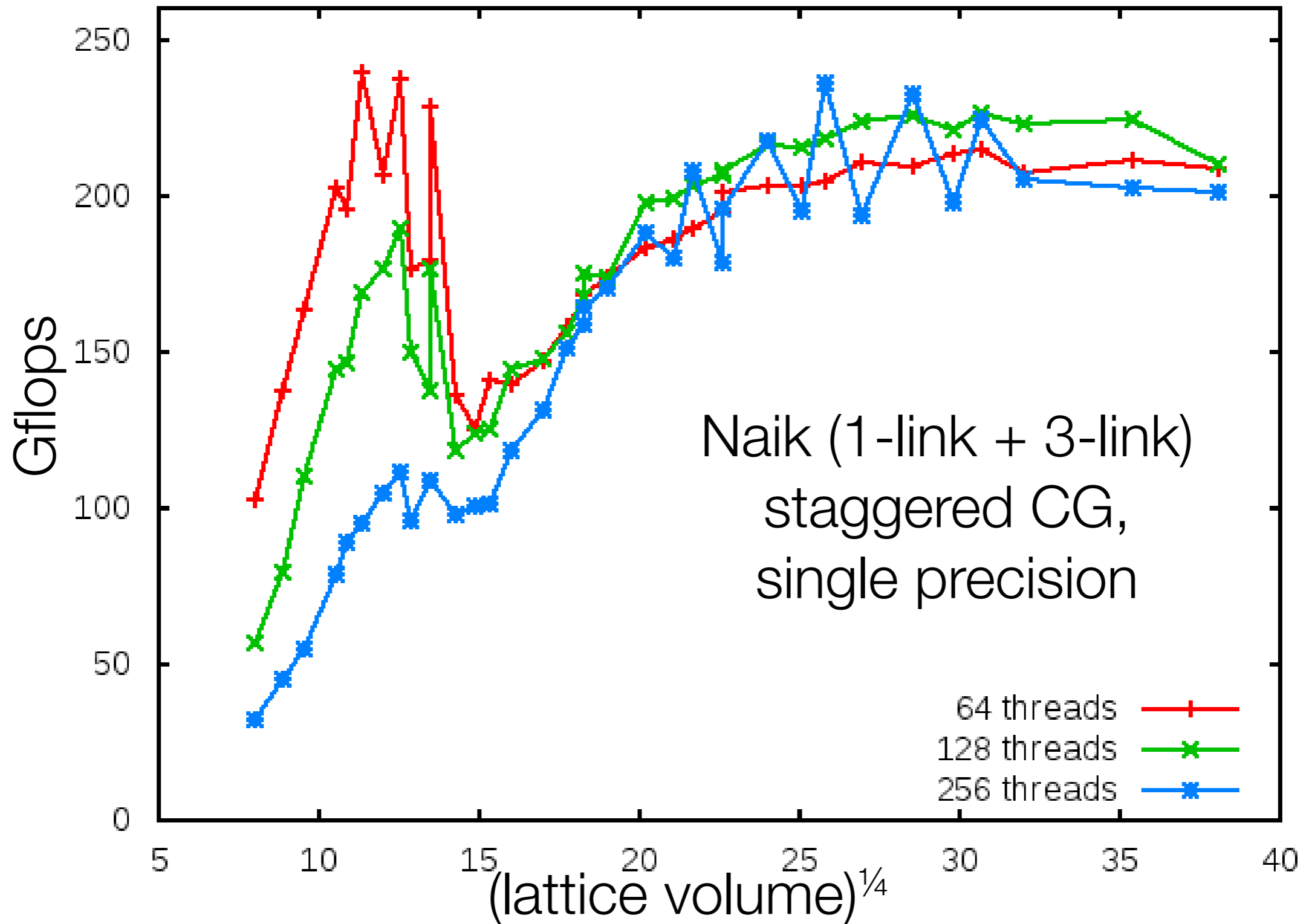
---

- Single node KNL
- Intel Xeon Phi CPU 7210
  - 64 cores, 4 hardware threads/core, 16 GB high bandwidth memory
- Benchmark staggered CG (w/ & w/o Naik term)
- Volumes  $L^3 \times T$ 
  - $L \in \{8, 12, 16, 24, 32\}$ ,  $T \in \{8, 12, 16, 24, 32, 48, 64\}$
  - with 64, 128, 256 OMP threads
- gcc 6.1

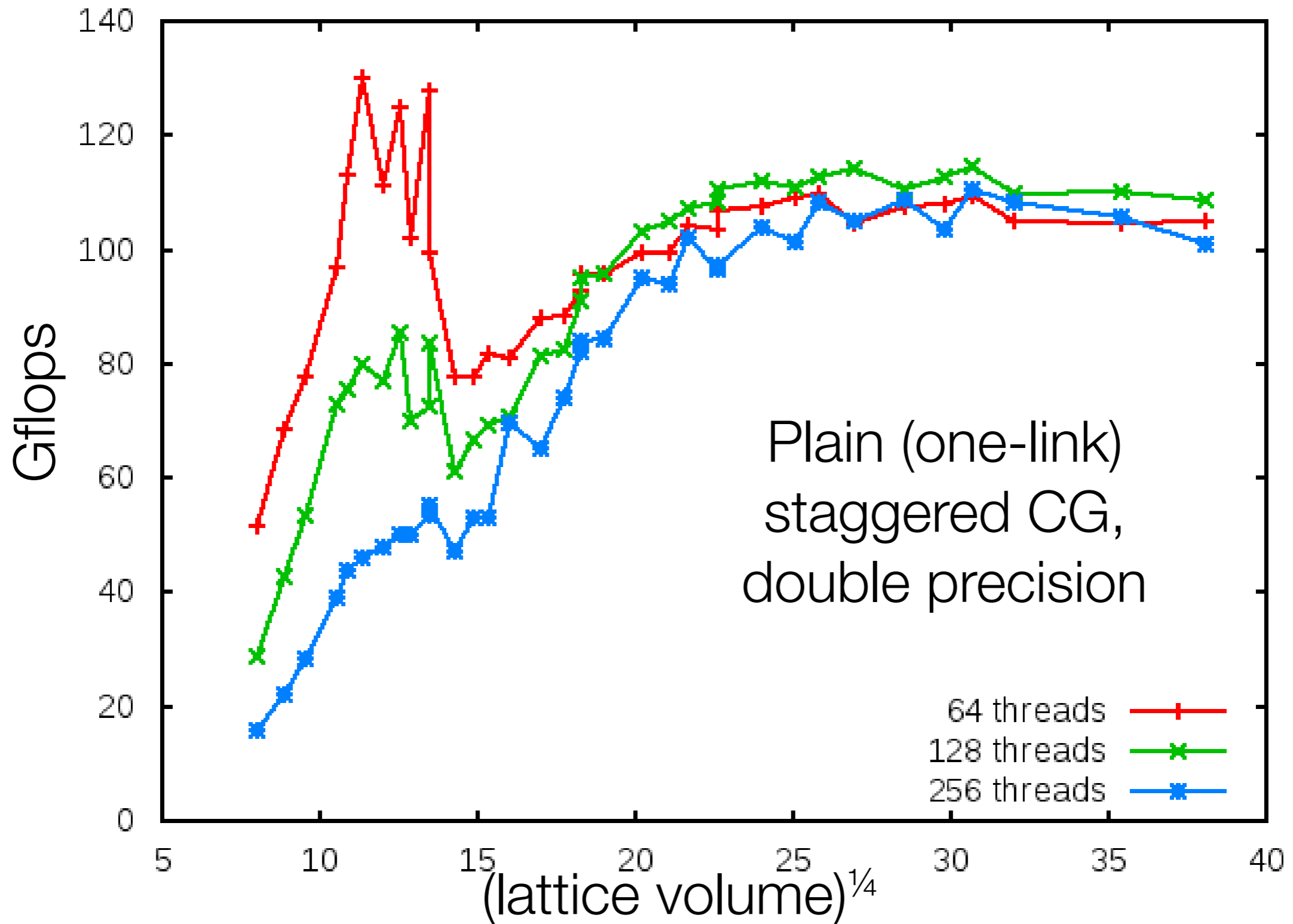
# QEX — Performance on KNL



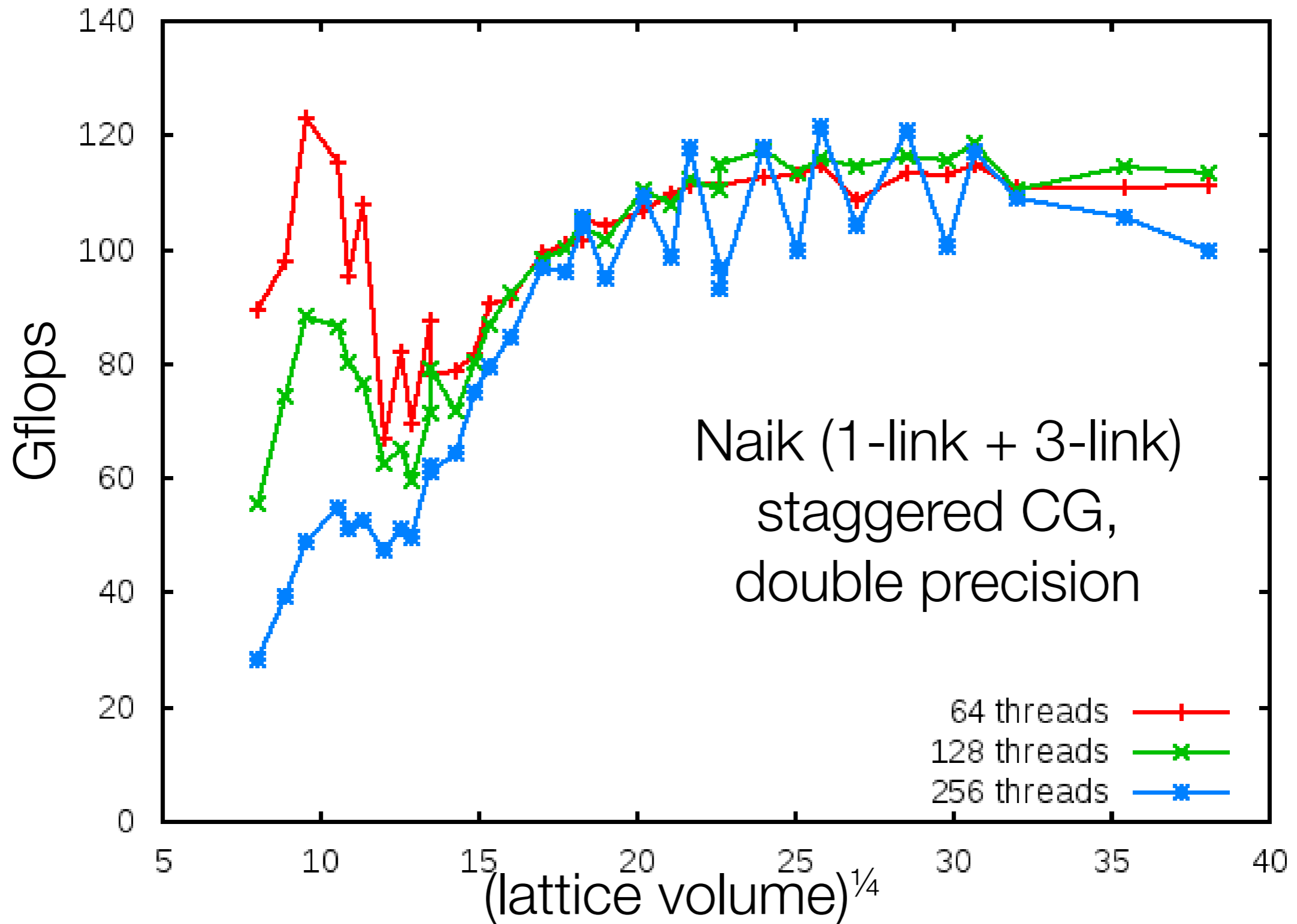
# QEX — Performance on KNL



# QEX — Performance on KNL



# QEX — Performance on KNL



# Summary

---

- Nim offers essential features for HPC
  - Extensive meta-programming with flexible syntax
  - Integrated and fast build system with modules
  - Seamless integration with C/C++ code, intrinsics, pragmas, etc.
- New QEX framework written in Nim
  - Staggered CG running with good performance on x86 (BG/Q in progress)
  - Working on optimization frameworks across compilers & architectures
  - Find more ways to exploit meta-programming to create easy to use DSL for specific operations: smearing, operator contraction
  - Synthesize reusable modules/libraries for other fields