Gauge field generation on large-scale GPU-enabled systems

Frank Winter

University of Edinburgh

The 30th International Symposium on Lattice Field Theory
Cairns Convention Centre, Cairns, Australia
Sunday, June 24 - Friday, June 29

Outline

1. Motivation
2. QDP-JIT
3. Demonstration HMC on TitanDev
4. Clover Comment / Future Work

Table of Contents

Motivation

- Want to use Chroma on large-scale GPU-enabled systems
- Titan/GPU clusters
- Gauge configuration generation
- Analysis of configurations

- Jaguar upgrade (Titan)
- 18.688 Cray XK6 blades
- Each blade comprises:
  1 NVIDIA X2090
  1 AMD Interlagos
- 768 GPUs already available as TitanDev

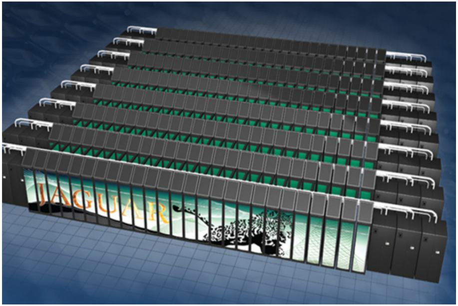


Table of Contents

QDP-JIT: Overview

- QDP-JIT provides 100% QDP++ implementation for GPUs
  - Automatic off-loading of single expressions to accelerators
  - Just-In-Time (JIT) compilation using NVIDIA NVCC as the jit-compiler
  - Automatic memory management via a caching mechanism
- Improvements since last year:
  - Parallel architecture support
  - Automatic tuning of CUDA kernels
  - Supports global reductions, arbitrary Sets on the GPUs
  - Improved overall performance

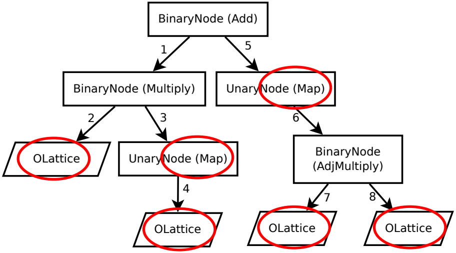
QDP-JIT: Parallelization Strategy

Outer	× Spin	× Color	× Reality
Scalar	Scalar	Scalar	Real
Lattice	Vector	Vector	Complex
	Matrix	Matrix	
		Seed	

- CUDA Thread parallelization on QDP++ outer level
- Each lattice site assigned to different CUDA thread
- Outer level absent in GPU kernel code
- Other levels preserved in kernel code

QDP-JIT: Just-In-Time Compilation

- Tree parser implemented with PETE
  - Traverses C++ type representing the expression
  - Caches and locks data objects
  - Generates CUDA C++ kernel code (uses QDP++ modulo outer level)
- JIT compilation (`system()` call)
  - NVIDIA NVCC builds CUDA kernel as shared object
  - `.so` loaded via `dlopen`() call
  - Kernels from prior runs loaded upon program startup



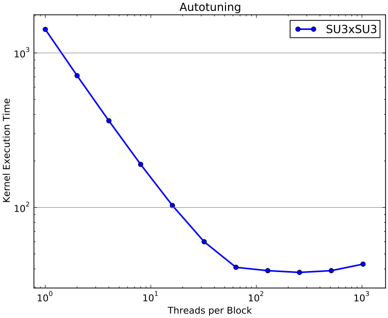

```
__global__ void kernel(kernel_args_t
     * args)
{
  int idx = threadIdx.x;
  OpAssign(
    ((ColorMatrix*)(args.ptr0))[idx]
  OpMultiply(
    ((ColorMatrix*)(args.ptr1))[idx]
    ((ColorMatrix*)(args.ptr2))[idx]
  );
}
```

QDP-JIT: Automatic Memory Management (Software Cache)

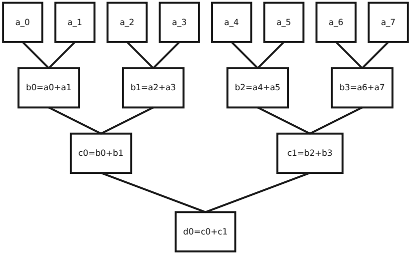
- Memory management completely reworked within QDP-JIT
- Cache is introduced
- Objects on the outer level
  Construction: register at the cache (memory allocation deferred)
  Destruction: sign off (decoupled from memory deallocation)
- Spilling algorithm: Least Recently Used
- Advantages:
- Memory is allocated upon first access (instead of at creation time)
  Allocation either on CPU or GPU
  Relaxes memory constraints for CPU
- Enables asynchronous kernel execution:
  Overlapping of GPU computation and communications
  Overlapping of GPU computation with CPU computation
- Be careful: C++ scoping!

QDP-JIT: Automatic Tuning of CUDA Kernels

- Free parameter within kernel launch: Threads per block
- Kernels generated dynamically:
  Using fixed value for all kernels results in non-optimal performance
  Optimal value cannot be predicted
- Measure kernel execution time as function of threads per block
- Store optimal value in XML database
- DB Key: Pair(kernel,local volume)



QDP-JIT: Global Reductions

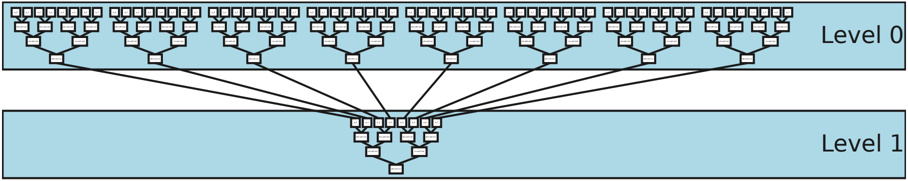


- E.g. `sum()`, `sumMulti()`, etc.
- Intra-node:

  Reduction in GPU memory:
  Tree-based approach used within each thread block (in DP)
- Inter-node:

  Spill final element to host memory
  Call-out to MPI reduce

- Avoid global sync by decomposing into multiple kernel invocations



QDP-JIT: Interoperability with QUDA

- QUDA (Krylov space solver package for NVIDIA GPUs)
- Interoperates with QDP-JIT memory management
  QUDA's device allocation redirected to cache allocation

Matrix inversion

QUDA

Solver solution reconstruction
Pseudofermion refresh
Momenta update
Quark smearing
Contractions
Force terms
Projection of fields
Leapfrog integration
Reunitarization (SU3)
Random number generator
Exponentials of color matrices
...

QDP-JIT

Table of Contents

Demonstration: Running on TitanDev

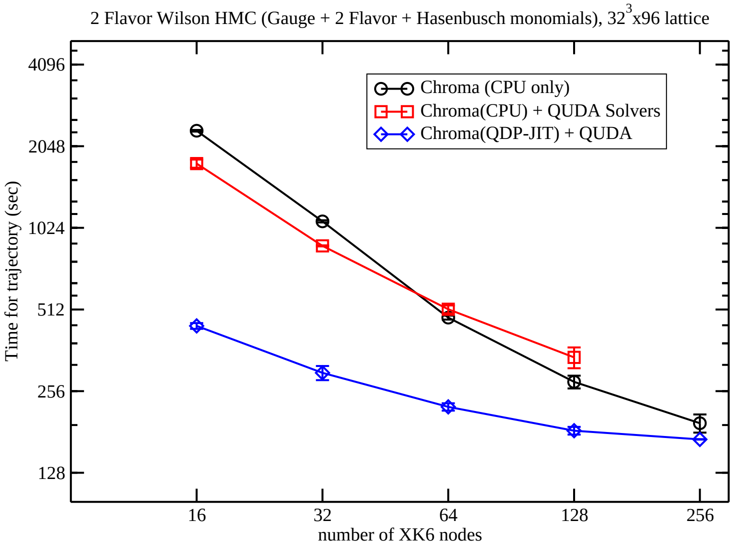
- HMC, Pure Wilson, $32^3 \times 96$, $\kappa = 0.13928$, $\beta = 5.5$
- Multi-timescale Integrators
- Chronological Inverter by Minimal Residual Extrapolation
- Gauge + 2 Flavor + Hasenbusch monomials

  $\rightarrow$ Total: 183 CUDA kernels

- Cray XK6 blades with NVIDIA X2090
  - Cray Linux Environment
  - No JIT (limited C library)
  - But `dlopen()` on compute nodes

    $\rightarrow$ Load pre-compiled kernels

Demonstration: HMC on TitanDev



- Data from TitanDev at OLCF, B. Joo & F. Winter

Table of Contents

Clover Status

- Clover is "broken out" of QDP++
  Internally stored as 2 triangular matrices
  $\rightarrow$ Layout-specific implementation

- Status:

  All kernels are developed for the GPU:
  Clover creation, inversion, application, packing for Quda
  Stouting kernels (Force term)

- Integration ready soon

Future Work

- Improving raw performance
  Direct peer-to-peer copy with CUDA 5.0 and OpenMPI 1.8
- Further integration with QUDA:
  Linear operators (Hasenbusch forces/Chronological solver)
  GPU pointers (No host-device transfers)

- Summary:
  QDP-JIT introduced
  HMC (pure Wilson) on Titan(Dev)
  Clover coming soon

- `git://git.jlab.org/pub/lattice/usqcd/qdp-jit.git`