Qthreads: A Library for Lightweight Threading

Stephen Olivier
Center for Computing Research

PSAAP WEST Workshop
January 27, 2016
What is Qthreads

- **Qthreads** is a user-level library for lightweight multithreading on the node, accessible directly using a C API or indirectly through a variety of higher-level programming models.

- **Initial motivation**: Support graph processing / analytics applications on commodity hardware using runtime-managed massive multithreading and rich synchronization primitives.

- **Current direction**: Expand workload focus to include computational science applications and numerical libraries while sustaining our graph / analytics support.
Qthreads Objectives and Outcomes

- Be efficient for both analytics and computational science
- Be a vehicle for run time system research
  - Enable co-design leveraging Sandia expertise across the system stack
  - Use modularity for flexibility, extensibility
    - Support for diverse architectures and programming models
    - Interfaces to inter-node technologies (MPI, Portals, distributed AMT)
- Improve understanding of system and application behavior
  - Runtime system introspection
- Impact deployed run time system technologies
  - Develop solutions to unsolved problems in adaptive run time systems
  - Engage with industry because our resources are limited
    - Vendors apply our techniques to their implementations, e.g., OpenMP
    - In Chapel language, our run time serves as Cray’s on-node tasking layer
Qthreads System Model

- The programmer exposes application parallelism as massive numbers of lightweight tasks (qthreads).
  - Problem-centric rather than processor-centric work decomposition to enhance productivity with transparent scaling
    - Both loop-based and task-based parallelism supported
  - Full/empty bit primitives for flexible, lightweight synchronization
    - Emulates behavior of Cray XMT (ThreadStorm) architecture
  - C API with no special compiler support required

- The run time system dynamically manages the scheduling of tasks for locality and scalable performance.
  - Heavyweight worker pthreads to execute the user’s tasks
    - Worker pthreads pinned onto underlying hardware cores
    - Architecture-aware mapping of workers to hardware (e.g., NUMA or Phi)
  - Lightweight task switching
Qthreads-based Software

- Chapel programming language
  - Chapel is Cray’s next generation parallel programming language.
  - Qthreads provides the default tasking layer for the Chapel runtime.
- Multithreaded graph library (MTGL)
  - MTGL is Sandia’s toolkit for graph processing algorithms.
  - Qthreads allows MTGL, originally designed for the Cray XMT, to execute on commodity systems such as x86 and POWER machines.
- Kokkos task-parallel extensions [*Experimental, maturing*]
  - Kokkos is Sandia’s C++ library for efficient management of data layout and parallelism for manycore processors.
  - Qthreads supports extensions to Kokkos that add task parallelism (futures) to the existing Kokkos data parallel capabilities.
- Qthreads has also been an OpenMP runtime using the LLNL ROSE compiler as the front-end. [*Not currently funded work*]
Recent Research Highlights

- **Task parallel over-decomposition**
  - Converts bulk synchronous codes to task parallelism and enables Qthreads to switch between computation and communication tasks
    - Barrett et al., *PMAM 2015*
    - Stark et al., *ExaMPI 2014*

- **Concurrency throttling for power savings**
  - Reduces the thread count and powers down cores when memory saturation is detected through CPU performance counters
    - Grant et al., *HP-PAC 2014*
    - Porterfield et al., *HP-PAC 2013*
Task Parallel Over-Decomposition

- Demonstration using Sandia’s miniGhost ghost cell mini-app
  - MPI (funneled mode) with communication-aware Qthreads scheduling
  - Stencil code encapsulated as tasks (minimal changes to application)
  - Suggests task parallelism may be a common base for HPC and analytics on a future unified architecture with high network injection rates

\[
\begin{aligned}
\n\text{u}_0 & = 0 \\
\n(u_{xx} + u_{yy}) & = f \\
\end{aligned}
\]
Code modification for miniGhost

Data parallel model:

```c
int stencil (...) {
    Exchange_boundary_data ( ... );
    Apply_boundary_conditions ( ... );
    Apply_stencil ( ... );
}
```

Task parallel model: Loop over blocks; spawned code is the usual data parallel model.

```c
ierr = MG_Block_init ( blks, ... );

for ( i=0; i<numblks; i++ ) {
    spawn ( stencil ( i, ... ));
}
```
Model comparison

BSP model

communication
computation

Task parallel model goal

Goal

t_i

t_j

time
Over-Decomposition Results

[PMAM15 and ExaMPI14 (Barrett et al., Stark et al.)]
Dynamic Concurrency Throttling

- Demonstration using LLNL’s LULESH hydrodynamics mini-app
  - OpenMP+MPI hybrid code using Qthreads as the OpenMP run time
    - Unmodified MPI across 27 nodes, each with 16 cores
    - Independent Qthreads instance on each node monitors power, memory
  - Power savings when the run time system throttles to 12 cores
  - Relief of memory pressure improves performance versus 16-core runs

[HP-PAC14 and HP-PAC13 (Grant et al., Porterfield et al.)]
Contributors to Qthreads R&D

- Richard Barrett, Carter Edwards, Ryan Grant, Courtenay Vaughan, Kevin Pedretti, Jon Berry, Siva Rajamanickam (SNL)
- Kyle Wheeler and Rich Murphy (now at Micron)
- Dylan Stark (now at Allstate)
- Brian Barrett (now at Amazon)
- George Stelle (UNM)
- Allan Porterfield and Jan Prins (UNC/RENCI)
- Brad Chamberlain and Greg Titus (Cray)
- Bronis de Supinski and Martin Schulz (LLNL)
- Marc Snir and Alex Brooks (UIUC)
- Alina and Dragos Sbirlea (Rice)
Qthreads Publications (1)


Available Online

Qthreads

http://www.github.com/Qthreads