The Chapel Tasking Layer Over Qthreads

Kyle B. Wheeler, Richard C. Murphy, Dylan Stark, and Bradford L. Chamberlain
The Structure of Chapel’s Runtime

Chapel Runtime Support Libraries
(written in C)

Tasks
Communication
Memory
Timers
Launchers
Standard
Chapel’s Tasking Layer

• **Role:** Responsible for parallelism/synchronization

• **Main Focus:**
  – support begin/cobegin/coforall statements
  – support synchronization variables

• **Main Features:**
  – Startup/Teardown
  – Singleton Tasks
  – Task Lists
  – Synchronization
  – Control
  – Queries
  – ...serialization?
The FIFO Tasking Implementation

• Work-queue model
  – Function calls for work execution
  – Centralized queue

• Pros:
  – Simple, easy to debug
  – Very portable
  – Uses native state management
    • stacks
    • thread/task-specific data

• Cons:
  – Task synchronization (sync) using thread synchronization (pthread_mutex_t)
    • Compute/synch overlap requires oversubscribing (#threads > #cpus)
    • Difficult to provide non-native (non-mutex) synchronization behavior
  – #Task-to-#thread mismatch creates unexpected deadlock potential
  – Does not support work stealing
  – Does not support CPU pinning
Challenges in Highly-Threaded Runtimes

• **Per-thread state**
  – State vs threads

• **Locality**
  – An afterthought in standard threading models
  – Communication and synchronization are expensive, easy to use accidentally

• **Synchronization**
  – Hard to make portable, maintain guarantees

• **Every Machine is Different**
  – Granularity of sharing (cacheline size)
  – Optimal number of threads (PU count)
  – Communication topology
  – Cache structure
  – Memory model
  – Synchronization Primitives (**CMPXCHG vs TNS vs CASXA vs LDARX/STWCX**)

Wednesday, May 18, 2011
Qthreads Highlights

• Lightweight User-level Threading (Tasking)
• Platform portability
  – IA32/64, AMD64, PPC32/64, SparcV9, SST, Tilera
  – Linux, BSD, Solaris, MacOSX
• Locality awareness
  – “Shepherd” as thread mobility domain & locality
• Fine-grained synchronization semantics
  – Full/Empty Bits (64-bit & 60-bit)
  – Mutexes
  – Atomic operations (Incr & CAS)
• Locality-aware Workstealing Model
Chapel Single Locale Challenges

• Startup & Teardown
  – Functions with unspecified scope
  – Synchronization primitives of unspecified scope

• Unsupported Behavior
  – Limit on OS Threads
    • Default defined by hardware
  – Forced serialization of tasks
  – Task-local data
Chapel Multi-Locale Challenges

• Communication (via GASNet)
  – Blocking system calls
    • Dedicated OS thread
    • Possibility for proxying internally
    • Temporary solution: Forked initialization thread
    • Future solution: explicit progress thread creation
  – External Task Operations
    • Task creation from outside the task library
      – Memory management issue
      – Also: synchronization issue…
    • Task synchronization outside the task library
      – Proxy-task using thread-level synchronization (pthread_mutex_t)
Future Work

• **Synchronization**
  – Tasking interface assumes only mutex semantics
  – MTA/Qthreads interface provide fast FEB semantics
  – Implementing FEB semantics with a mutex implemented with FEB operations is silly and slow

• **Stack Space**
  – Problem common to all tasking interfaces
  – Currently requires guess-and-check
  – Potential directions:
    • Technically possible to calculate stack requirements (e.g. gcc 4.6)
    • Technically possible to move stack variables to heap
      – Moves the memory management problem
Performance: Raw Tasking

- **QuickSort**
  - Naïve implementation (serial partitioning)
  - Uses recursive \texttt{cobegin}
  - Serialization threshold
    - For best comparison, set high to avoid serialization

![Graph showing execution time and ratio]

- **Execution Time (secs)**
  - Array Elements (power of 2)
  - Ratio FIFO/Qt

- **Qthreads**
- **FIFO**
Performance: Raw Tasking

• Tree Exploration
  – Constructs binary tree
  – Assigns Unique ID
  – Computes sum of IDs
  – Uses recursive `cobegin`
Performance: Data Parallel

• **HPCC RandomAccess**
  – GUPS (random integer updates)
  – Stresses Memory System
  – Uses `forall`

![Graph showing execution time vs. number of tasks for Qthreads and FIFO methods.](image)

Wednesday, May 18, 2011
Performance: Data Parallel

• HPCC STREAM (-EP)
  – Memory Bandwidth & Vector Kernels
  – EP version avoids communication
  – Uses `forall`
  – Synchronization surprisingly important
Thank You!

Questions?