Xpress Performance Workshop
RCRblackboard

Allan Porterfield
RENCI, UNC-Chapel Hill

Rob Fowler
RENCI, UNC-Chapel Hill
Why RCRToolkit?

- Thread performance on multi-core systems limited by what the other threads on the system are simultaneously doing.
  - L3 cache contention
  - Memory bandwidth limitations – includes contention in DIMMs
  - Internal bus contention

- RENCI has been working on Resource Centric Reflection toolkit to expose contention to programmer and runtime.
  - Performance Tuning Tools (RCRoolkit)
  - Runtime Scheduler (MAESTRO scheduler in Qthreads)
Pieces of RCRToolkit

• Infrastructure
  – RCRblackboard – database to store dynamic information about system
  – RCRdaemon - allow user access to hardware performance counters
  – RCRlogger – allow post-execution review of counters
  – RCRviewer – simple GUI to view results

• Clients
  – EnergyStat API – allow user to see energy application required
  – Qthread scheduler – allow runtime access to dynamic information
  – HPCToolkit – Modified to query blackboard
Jobs
(RCR API / Qthreads)

Other Performance Tools (HPCToolkit etc.) / Power Control Tools

RCR Toolkit

RCR Logger

RCR Blackboard

RCR Daemon

RCR Viewer

App 1

App 2

App 3

RCR Toolkit

Libra – like output

Network

Rack 1

System 0

Shared NICs

IPMI / RAS

Socket 0

Core 0

Core 1

Core 2

Core N

Shared Caches

Shared Resource

Node 0

Node 1

Rack 0

Rack 1

Jobs

RCR Blackboard

RCR Daemon

RCR Viewer

RCR Logger

Other Performance Tools (HPCToolkit etc.) / Power Control Tools

(RCR API / Qthreads)
RCRblackboard (1)

- Publisher/Reader Semantics
  - Each section 1 writer multiple readers – eliminate synchronization
  - No reader checkin – writer does not produce events for readers
  - Self-describing data format that writer/readers agree on

- Uses shared memory regions
  - One per writer
    - currently only one writer – it uses /dev/shm/bbFile
**RCRblackboard (2)**

- **Google Protobuf**
  - Self-describing, compact
  - Seems designed for network and stores in a compressed format
    - Compression on every write is very expensive for us
    - Future – write store function that doesn’t compress
      - Updates become simple writes / no compression
      - Reads are simple reads / no expansion
      - Will need to define mechanism to prevent inconsistent data being read
        (when reading multiple values – 2 version numbers?)
  - Hierarchical based on classes from protoc
    - On our 2 socket SandyBridge system
      - 8 sets of core counters
      - 2 sets of socket counters
      - System-wide counters
Message RCRBlackboard {
    optional RCRBlackboardMetadata bbMetadata = 1
    repeated RCRNode node = 2
    repeated RCRSocket socket = 3
    repeated RCRCore core = 4
    repeated RCRSocketMeter socketMeter = 5
    repeated RCRCoreMeter coreMeter = 6
}
RCRdaemon

- **Write hardware counters into RCRblackboard**
  - Chip-wide – energy/L3 cache/Memory Controller
  - Core-specific – std set (cycle cnt/floating pt/etc.)
- **Several Architecture specific versions**
  - Intel SandyBridge (currently used)
  - Intel Nahalem (compiles – as of Monday / untested – doesn’t crash immediately)
  - AMD Opteron (used in the past and probably victim of bit rot)
- **Needs to run at kernel protection level to access global counters**
  - Configuration dependent (energy counter requires it / as do some L3 counters)
- **Writes /dev/shm/bbFile using protobuf interface**
  - Current overhead ~16% of one core
  - Big savings by eliminating compression (one per write)
RCRlogger

• Reads RCRblackboard periodically and writes results to stdout
  – Dumps all active counters on single line
  – Identified by socket/core number and counter number
  – Up to ~12000 times a sec on Intel SandyBridge (2.7GHz)
    • Faster than many of the counters update in RCRblackboard (energy ~1000)
  – Startup option to set frequency (-i # in microsecs)
  – -d turns into daemon (no stdout – not sure why)
  – No -f output to filename (should be added)
RCRviewer
EnergyStat API

- Provides a pair of calls (in C) to capture energy usage during a program
  - extern “C” int energyDaemonInit(int wait);
  - extern “C” void energyDaemonTerm();

- Produces these lines of output
  - (init call) Starting doEnergyWork
  - (term call) Application (Energy) – Time 8.109619 Total energy consumed 1072.728810 Ave. Power Level 132.278572 Final Temperature socket 1 – 53.000000 socket 2 – 46.000000

- Multiple calls to energyDaemonTerm allowed
  - Each prints energy since previous call

- Initiates low-overhead daemon
  - Wakes up every wait nanoseconds and reads counters (32 bit – protects from overflow)
  - Only works on Intel SandyBridge (and probably IvyBridge) processor
Sherwood Scheduler

• Locality-aware scheduler for Qthreads
  – Work sharing between cores sharing L3 cache
  – Work stealing between sockets sharing an address space

• Modified to reduce energy consumption
  – Reads energy and memory concurrency from RCRblackboard
  – If both high reduces the number of active threads
    • Duty-cycle modification to greatly reduce power requirements of idle threads
  – Saved ~3% power for benchmarks/Mini-Apps where it applied
    • Micro-algorithm benchmarks(UNC), BOTS suite(Barcelona), and LULESH (LLNL mini-app)
HPCToolkit hot-wired with RCRToolkit