Xpress Performance Workshop
– The Future of RCRblackboard
XpressBlackboard

Allan Porterfield
RENCI, UNC-Chapel Hill

Rob Fowler
RENCI, UNC-Chapel Hill
RENCI/UNC’s Version of the EXPRESS goal

• Provide a complete slice of a software stack that allows unified approaches to the difficult problems facing Exascale software
  – Massive parallelism
  – Heterogeneous parallelism
  – Resilience
  – Reliability
  – Power management
  – Performance
  – Unified software tools
XPRESS must be a single software stack

- XPRESS must provide a unified software stack
  - Not a set of independent utilities
    - Attack Exascale problems
      - Resilience/Reliability at all levels
      - Billion-way Parallelism
      - Heterogeneity
      - Control Energy Usage
    - Inter-operate smoothly
    - Information flows across boundaries within the stack
      - How is the system performing?
      - How are the other tools responding to the current performance?
- DOE wants the entire X-stack tool chain to be unified
  - Not just XPRESS
XPRESS information

• Passing information between utilities will be critical
  – In current systems information flows one direction
    Application -> Compiler -> Runtime -> OS -> Hardware

• For Exascale static scheduling decisions will not work
  – Dynamic environment
    • Billion-way parallelism
    • Resilience
    • Reliability
    • Energy
    • Shared resource contention

Feedback will be required
Exascale Performance Information flow

- Application
  - ROSE
    - OpenMP
      - HPX-4
- Application
  - LLVM
    - MPI, TAU
      - HPX-4
- Performance Tuning
  - Compiler
    - Libraries
      - HPX-4

XpressBlackboard

Exascale System
Performance Information as Glue

- **Performance information**
  - Current – post-execution performance tools
  - Exascale – dynamic application introspection

- For performance and reliability thread/core/node/system knowledge will be critical throughout the software stack
  - Interfaces designed to enable information flow
    - Utilities need to know current system performance
    - Utilities need to know how other utilities are reacting
XpressBlackboard Structure

• Maintain write integrity by only allowing 1 writer per region
  – Multiple readers – Publisher/Reader model
    • Don’t have to register – No knowledge of who is currently reading data
  – Structure
    • Each section starts on page boundary
    • 1st page - Index to locate relevant pages
    • 2nd page - first writers data
      – Readers and Writer agree on format
        » Self-describing – like protobuf
    • Following pages – more writers

• Local
  – Data specific to a particular locale
XpressBlackboard API

- **Register new writer/reader**
  - void *AcquireRegion(int Type, int Length);
    - Each writer needs to allocate region
  - int ReleaseRegion(int Type);
    - Only writer can release – returns 0 if successful
  - void *FindRegion(int Type, int *length);
    - Reader locates desired shared memory

- **Read/Write**
  - Standard memory operations
Global XpressBlackboard

• Not all information available locally
  – Global health and energy
  – Dynamic load balance

• Replicating information not practical
  – Million copies too big
  – Uses too much network
  – Global coherence too hard

• Global address space means reachable
  – Single copy possible
  – But are network costs (latency and bandwidth) acceptable?
Global XpressBlackboard

• Hierarchical Tree
  – Global data – with well-defined compaction methods defined

• Each level has 1 daemon
  – Keeps table of values from children
  – Communicates single value to parent
  – Receives global value from parent
  – Transmits to children
    • Write into blackboard of child in same address space
    • Trigger event in child to recognize new value
    • Could be message when child not a leaf
Global XpressBlackboard API

- **Register New Global Data**
  - int RegisterGlobal(Level h, CompactionType t, int freq, int *offset)
    - Setup tree for this global value and return id – done once
      - Level – what region of system is this value visible (system, row, rack)
      - Type – what rule is to be applied for data compaction (max, min, ave.)
      - Freq – how often does the daemon update the global value
      - Offset (?)- offset into the local XpressBlackboard region result
  - int DestroyGlobal(int id, Level h)
    - Remove global no longer required
  - void WriteGlobal(int X, value)
    - Locally produce a new value for global variable X
  - Read using standard memory op returned offset
Global XpressBlackboard API

• **Expected Uses**
  – Will be used for system data – temperature, energy
    • Setup by LXX – never deleted
  – Will be used for application or workflow specific data – progress, load balance
    • Setup by application or scheduler – deleted on termination

• **Concerns**
  – Daemon need to be able to add and delete globals
  – Daemon needs strategy for missing (or very old data)
  – Needs to be very light-weight but still allow variable frequencies (maybe fixed set)
  – Data supplied after global removed
XpressBlackboard Use Case – Detect Resource Contention - Hardware Counters/TAU

• First Test – hopefully can prototype this week
• RCRdaemon on SandyBridge (IvyBridge?) systems writes into blackboard ~1000 times a second
  – energy (socket) MSR_PKG_ENERGY_STATUS
  – memory occupancy (core) cX_MSR_PMON_CTL1 – TOR_OCCUPANCY
• Use TAU to identify high/low power regions of the code
  – Looks good in demo – what the programmer will do with this information?
• Use TAU to identify regions where overall occupancy approaches limits
  – Identify memory bottlenecks
    • User rework the application to spread out memory accesses
    • Rework to reduce parallelism with little no effect (improved cache utilization?)
XpressBlackboard Use Case – Load Balance Information – Application to HPX or L XK (1 of 2)

• Detect load imbalance
  – Thread waiting times - PMPI writes wait time into blackboard compute global average – locally check to see if “near” average
    • Much higher – too idle
    • Much lower – too busy
  – Lightweight thread/task queue occupancy – HPX updates blackboard – compute global maximum
    • Max high – local near zero – idle
    • Max high == local value - busy
  – CPU Utilization – global average
    • Global High – local near zero – idle
    • Global Low – local high - busy
• Load imbalance decisions
  – Make sure busy node running at correct power state (LXK/HPX)
  – Set idle node to low power state (LXK/HPX)
  – Rebalance work – move work to idle nodes from busy nodes (App/HPX)
    • Increase amount of work in each thread on idle nodes
    • Move work off busy nodes
    • Increase priority to schedule “slow” threads earlier

• Need to communicate who is doing load balance
  – Don’t want to move work in application and HPX and just make a new overloaded node – which LXK has just slowed down too save energy
XpressBlackboard Use Case – Thread Scheduling – Performance Counters to HPX

• **Goal**: produce answer in same time using less energy

• **Detect Resource Contention**
  - TOR_OCCUPANCY (summed counters near known max)
  - NIC utilization (above a threshold)
  - Memory allocation (detected either inside app or by L XK)

• **Adjust active parallelism to limit resource contention**
  - Can limit threads assigned work (adjust duty cycle of idle threads)

• **Slow entire node to limit pressure on resource**
  - Use DVFS to set node in a lower power state
    - Only works if resource not effected (ie DIMMs speed not effected by processor power state)
XpressBlackboard Use Case – Health Information – IPMI counters to L XK

- Use global counters to detect health anomaly
  - Excessive ECC errors
  - Excessive Temperatures
  - Fan speeds
  - Unusual power requirements (tricky)

- Adjust work
  - Migrate off failing processors
  - Remove processor from allocate list until issue resolved
XpressBlackboard Use Case – Phase Specific Execution Power State – MPI to LXK/HPX

• MPI DVFS research has found many application where energy saving can be accomplished by reducing power for specific phases

• MPI use the blackboard to mark phase boundaries
  • LXK use to control power
  • HPX use to improve scheduling
  • APEX/TAU use to improve program attribution

• Application or Compiler could also identify phases
  • Automation increases usability/may decrease accuracy