The Open Community Runtime
A Runtime System for Extreme Scale Computing

OCR Team – Vincent Cavé (Intel)
WEST Workshop 02/2017

Acknowledgment: This material is based upon work supported by the Department of Energy Office of Science under cooperative agreement DE-SC0008717.
Outline

• OCR 101
• Runtime Overview
• Experimental results
Context: An Exascale System

• Large core count, low power
• De facto heterogeneity
• Deep memory hierarchy
• Mean time between failures will be short

• Approach problem through a HW/SW co-design
Open Community Runtime (OCR)

• An Asynchronous Many Task (AMT) model
• OCR Specification (v1.1.0) & Execution model
• OCR Philosophy
  • Explicitly parallel
    • Task based, event driven, explicit dependences and relocatable data
    • Globally Unique Identifiers (GUIDs)
  • Runtime handles scheduling, load-balancing and resilience
    • Exploiting parallelism structure
    • Introspection
    • User-provided Hints
• Reference implementation in C
OCR 101

Dependences (Pre-slots)

OCR object

GUID

Post-slot

EDT  

Event

Datablock

Dependence; data not yet available

Dependence; data available

=> Dynamically build a Directed Acyclic Graph (DAG) of the program
Putting it together

Dynamically build a Directed Acyclic Graph (DAG) of the program

EDT

Event

Data block

Dependence: data not yet available

Dependence “satisfied”: data available
Data Abstraction: Data block

- Relocatable block of memory
- Acquire/Release model
  - Dependence defines access mode
  - Acquire transforms GUID into a “C” pointer
  - Pointer lifespan limited to the task
- Only data accessible to the task
- Release relinquishes access

```
dbPtrA[i]=42;
```

Single post-slot always satisfied

<< Acquire >>
<< Release >>
Synchronization Abstraction: Events

- Asynchronous control and data dependence
- Simple, Scalable and Composable
- Four kinds of events:
  - Sticky, idempotent, once, latch
- Trigger function
- Output function

1 or more pre-slots

Single post-slot

Dependence; data available

Event
Task Abstraction: Event Driven Task (EDTs)

- Non-blocking execution
- Self-contained (code+data)
- Side-effect free / replay-able
- Use events for synchronization
- All data known before execution
- Release consistency model

**EDT State**

- 0 or more pre-slots
- \(\ll<\) Created \(\gg\) \(\ll<\) Resolved \(\gg\) \(\ll<\) Runnable \(\gg\) \(\ll<\) Running \(\gg\) \(\ll<\) Triggered \(\gg\)

**Single post-slot**

**Application code + OCR API calls**

**EDT \(\ll<\) \(\gg\) State**

- Dependence; data not yet available
- Dependence; data available
Task/Data Distribution

• Ideally the runtime figures out
  • Default distribution policy

• Affinity hints
  • Query number of affinities
  • Use hint when creating EDTs/DBs

• Dispersion hints
  • FAR
  • NEAR

• Still more work to be done
OCR APIs Overview - Simplified

• EDT
  • Task templates: ocrEdtTemplateCreate(...), ocrEdtTemplateDestroy(...)
  • Tasks: ocrEdtCreate(...), ocrEdtDestroy(...)
  • Signature: ocrGuid_t edtName(u32 paramc, u64 * paramv, u64 depc, ocrEdtDep_t * depv)

• DBs
  • Management: ocrDbCreate(...), ocrDbDestroy(...)
  • Usage: ocrDbRelease(ocrGuid_t )

• Events
  • Management: ocrEventCreate(), ocrEventDestroy()
  • Satisfaction: ocrEventSatisfy(ocrGuid_t evt, ocrGuid_t db)
  • Dependence definition: ocrAddDependence(ocrGuid_t src, ocrGuid_t dst, mode)

• Miscellaneous
  • Entry point of OCR: mainEdt()
  • Shutdown: ocrShutdown(), ocrAbort()

• Working as part of HiHat to share API design learnings
OCR Reference Implementation

- Collective effort by Intel, Rice University and Reservoir Labs (06/2015)
- Compliant with OCR Specification v1.1.0
- Open source
- Supported platforms: x86, x86-mpi, x86-gasnet, TG
- Ongoing work and maturation
  - Introspection & Adaptation
  - Resilience
  - Performance
OCR Runtime Architecture
OCR Applications

- CoMD
- HPCG
- HPGMG
- XS/RS Bench
- Graph500
- Nekbone (wip)
- miniAMR (wip)
- Tempest (C++ - wip)

- Stencil 1D & 2D
- Cholesky
- Smith-Waterman
- FFT
- Fibonacci

- + Generated from Rice U. CnC and R-Stream
NERSC Edison Weak-scaling OCR wrt MPI

Known bottlenecks to address
Conclusion

• Define an execution model for Exascale
  • Targetable by High-Level languages
  • Explicitly parallel task, events, data
  • Maps to various platforms

• Community building
  • OCR specification
  • Open source reference implementation + other homegrown

• Encouraging performance results
OCR references

- Specification
  - [https://xstackwiki.modelado.org/Open_Community_Runtime](https://xstackwiki.modelado.org/Open_Community_Runtime)
- GIT repo reference implementation
  - [https://xstack.exascale-tech.com/git/public](https://xstack.exascale-tech.com/git/public)
- Wiki / Code Review / Continuous Integration
  - [https://xstack.exascale-tech.com](https://xstack.exascale-tech.com)
- Mailing lists
  - OCR-User
  - OCR-Dev

Questions ?