Task Characterization-driven Scheduling of Multiple Applications in a Task-based Runtime

Kavitha Chandrasekar
Bala Seshasayee
Ada Gavrilovska
Karsten Schwan
Motivation

• Why Multiple Applications?
  – To fully utilize large computation resources of Exascale systems
  – Common HPC scenario: Co-locating simulation and analytics
  – In-situ analytics aids in reducing inter-node data movement*
  – Reduced inter-node data movement => reduced energy consumption

• We focus on Exascale Runtime systems for serving HPC needs
  – They provide scalable framework and abstractions to control compute and data placement
  – Runtime Environment under consideration:
    • Single Runtime system on Exascale platform
    • No-OS solution to avoid cross-layer interaction costs

*Reference: Enabling In-situ Execution of Coupled Scientific Workflow on Multi-core Platform
Problem Statement

• Naïve co-location, however, can lead to
  • Shared resource contention
  • Poor resource utilization
• We evaluate the problem of
  • Co-scheduling multiple applications within a single task-based runtime
• Technical Contribution
  • Tight Integration of Resource management within the runtime
    • Support for Task-level characterization
    • Support for scheduling at Task granularity
Solution Approach: Runtime Tool

- Task-based Runtime for Future Exascale systems
  - Ability to *Scale*: Via highly Asynchronous tasks
  - Ability to *Tune*: Via abstractions for explicit control of compute and data placement
- Representative Framework: **Open Community Runtime** (OCR)*
- Underlying architecture: Experimental Exascale system, **Traleika Glacier** (more later)

*Link: [https://xstackwiki.modelado.org/Open_Community_Runtime](https://xstackwiki.modelado.org/Open_Community_Runtime)
Solution Approach: Task-Aware Scheduling

• Problem: Black-box techniques
  – Sampling happens at runtime system granularity
  – Use IPC, hardware counters
  – Leads to: Inability to decipher individual task characteristics

• Solution: Task-level characterization
  – Ability to distinguish task types
  – Aids in task-level scheduling decisions
Solution Approach: Task-Aware Scheduling

- Components
  - Hints Generation
  - Hints API
  - Scheduling heuristic
Hints Generation

• Hints Generation
  • Task characteristics/hints generated and propagated to scheduler
  • Types
    • Offline profiling
    • Online profiling
**Offline Profiling**

- **Byfl (Bytes/flop)** – LLVM-based tool to generate software counters
- Profiling run performed with Instrumented OCR application

- **Categorize tasks**
  - Compute intensive
  - Memory intensive
  - Cache intensive

<table>
<thead>
<tr>
<th>Metric &amp; Threshold</th>
<th>Task- categorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flops/Time &gt;5</td>
<td>Compute-intensive</td>
</tr>
<tr>
<td>(LD+ST)/Time &gt;5</td>
<td>Memory-intensive</td>
</tr>
<tr>
<td>(LD+ST)/flop &gt;5</td>
<td>Cache-intensive</td>
</tr>
</tbody>
</table>
Online Profiling - Discussion

Introspection module

EDTs
- Run ready EDTs in OCR
  - Spawn EDTs
  - Execute EDT
  - Update stored Hints

Worker
- Satisfy Event
- Repeat

Scheduler
- Schedule EDTs using hints

Feedback
- Process data
  - Start_profiling_event
  - End_profiling_event
Solution Approach: Task-Aware Scheduling

- Components
  - Hints Generation
  - Hints API
  - Scheduling heuristic
Hints API

u8 ocrTaggedEdtCreate( ocrGuid_t *guid, ocrGuid_t templateGuid,
                    ...
u8 edt_tag );

u8 ocrSetHint( ocrGuid_t guid, ocrHint_t *hint );

• Hints are passed to EDTs (Event-Driven Tasks)
• Hints can be passed at
  • Time of Task Creation
  • At runtime by triggering of events
• EDTs hints/tags are based on category
  • compute/memory/cache intensive
Solution Approach: Task-Aware Scheduling

- Components
  - Hints Generation
  - Hints API
  - Scheduling Heuristic
Scheduling Heuristic

• Scheduling in OCR on x86
  • Fine-grained scheduling at task-level granularity
  • Resource partitioning done via
    • Worker-partitioning
    • Task-worker mapping

• Scheduling in OCR on TG
  • Similar resource partitioning based on task characteristics
Experimental Evaluation

**Testbed**
- Intel Westmere
  - 2 hex-core sockets (12 virtual cores per socket)
  - 48 GB RAM (24 GB per socket)
- Exascale System
  - Traleika Glacier; Simulator: Fsim
  - Several low-power single-issue specialized execution engines (XEs)
  - Fewer x86-based control engines (CEs)
    - to orchestrate task scheduling, data mapping, and monitoring
Experimental Evaluation

- Benchmarks Used
  - Compute (synthetic)
  - Memory (synthetic)
  - CoMD (Molecular Dynamics)
  - Cholesky, Fibonacci
  - Stream
- Importance of Task-Characterization
  - Compared to homogenous tasks, heterogeneous tasks benefit from task characterization
  - Baseline – Task-unaware scheduling of homogenous tasks

**Figure 4:** Normalized improvement (percent) of homogenous versus heterogenous tasks' optimization potential.

**Figure 5:** Efficiency.

**Figure 6:** Performance.

**Table 2:** Applications Used.

<table>
<thead>
<tr>
<th>Application Type</th>
<th>Proxies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute (synthetic)</td>
<td>Simulation Compute, Cholesky, CoMD, Fibonacci</td>
</tr>
</tbody>
</table>

**Table 3:** Experimental Evaluation.

- **Benchmarks Used**
  - Compute (synthetic)
  - Memory (synthetic)
  - CoMD (Molecular Dynamics)
  - Cholesky, Fibonacci
  - Stream
- **Importance of Task-Characterization**
  - Compared to homogenous tasks, heterogeneous tasks benefit from task characterization
  - Baseline – Task-unaware scheduling of homogenous tasks
Task-aware Load-balancing on x86

- **Single Socket**
  - Reduced memory controller contention
  - Hiding memory access latency with task-mapping on hyper-threads
- **Dual Socket**
  - Increased bandwidth by distributing memory tasks across sockets
  - Maximizing memory bandwidth utilization
Energy Benefits on Exascale System

- Load-balancing of compute and memory intensive tasks
- With load-balancing, reduced contention at L2 level, hence fewer DRAM accesses
- DRAM energy usage is a dominant factor
- Task-aware scheduling reduces energy consumption by at least 50%
Related Work

• Auto-tuning
  • **PICS**: is a framework in Charm++ have a way to tune application parameters
    • Uses semantic knowledge of the application to steer decisions
  • **TreeMatch**: Takes into account per MPI process communication characteristics and topology in order to load-balance processes. Characterization is done at process level.

• Co-scheduling of multiple applications
  • **Goldrush**: simulation and analytics’ resource management based on application level tracking of performance counters and maintaining phase history
  • **Merlin**: Workload consolidation in datacenters; Resource allocation on multi-socket node by analyzing application sensitivities through online monitoring.

• Cross-layer Interactions:
  • **Performance points**: benefits of crosslayer APIs to achieve performance goals
Conclusion and Future Work

• Demonstrated importance of **task-level characterization** for **scheduling** of multiple co-located HPC applications
• Addressed methodologies for **task-profiling** including
  • Offline Profiling
  • Discussed approach for Online Profiling
• **Implementation using OCR**, a task-based runtime model, representative of programming models that befit future Exascale systems like Traleika Glacier
• Studied
  • **Performance** on existing x86 systems
  • **Energy benefits** on future Exascale system TG
• Future work
  • Online introspection
  • Compiler-based hints for profiling/characterization for introspection
Questions?