An Overview of GEO (Global Energy Optimization)

Project Lead:  Jonathan Eastep, PhD & Principal Engineer
jonathan.m.eastep@intel.com

December 9, 2015
GEO Project Scope and Goals

- GEO is an open source, scalable, extensible runtime and framework for power management in HPC systems
  - Provides extensibility via plug-ins + advanced default functionality
- Developing GEO through CORAL NRE project with potential deployment on Aurora system at Argonne
- Goal 1: unlock more performance in power-limited systems
- Goal 2: accelerate innovation in HPC power management
  - Enables researchers to focus effort on algorithms (via plug-ins) not re-engineering distributed runtime infrastructure
  - Provides a streamlined path for deploying new ideas
    - Product-grade framework w/ development+hardening backed by Intel
  - Drives codesign of power and performance management features in Intel processors for better results w/ runtimes like GEO
Acknowledgements

GEO Core Team (Intel)
- Fede Ardanaz
- Chris Cantalupo
- Jonathan Eastep
- Richard Greco
- Stephanie Labasan
- Steve Sylvester
- Reza Zamani
- ... and hiring!

Collaborators (Intel)
- David Lombard
- Tryggve Fossum
- Al Gara

Collaborators (External)
- Argonne (CORAL)
- LLNL (Rountree)
- ... and expanding!
Relationship to Standard Power APIs

- GEO is a job-level power management framework
  - Manages the compute nodes in a job to a job power bound
  - ... while maximizing performance or other objective functions
- With work, GEO could fit under/above other power APIs
  - GEO currently interacts with other SW components through its own interfaces (next slide)
  - We’re not positioning our external interfaces as standards
- Emphasis on providing an extensible framework and advanced out-of-the-box power management strategies
  - Builds on “Auto-Tuner” machine learning, control system, and optimization technology Intel has been researching for 4 years
GEO Interfaces / Integration Architecture

- **User Interface** (Work w/ RM & Schedulers)
- **Admin Interface** (Work w/ RM & Schedulers)
- **Resource Manager**
  - Scheduler = Power-Aware Scheduler (Work w/ Intel team to implement)
- **Job Power Manager Interface** (Work w/ RM & Schedulers)
- **Job Power Manager = GEO**
- **PCU RAPL and Perf Counter Interfaces** (Work w/ Intel GEO team to enhance)

**Owners**
- 3rd parties
- Intel GEO team
- Intel PM Arch team

**Application Interface**

Intel Corporation
Advanced Auto-Tuner Capabilities

- Comprehend and mitigate dynamic load imbalance by globally coordinating frequency and power allocations across nodes
- Leverage application-awareness and learning to recognize patterns in application (phases), then exploit patterns to optimize decisions
- React to phase changes at aggressive time scales (low milliseconds) and rapidly redistribute limited power to performance-critical resources
- Tackle the scale challenges prior techniques have swept under the rug to enable holistic joint optimization of power policy across the job
Auto-Tuner Prototype Results Summary

Speedup from Auto-Tuner at ISO Power

- **miniFE**: 1.18x
- **FFT**: 1.31x
- **IS**: 1.17x
- **NEKbone**: 1.22x

**Speedup derives from two factors:** correcting load imbalance across nodes and node-local spatio-temporal energy scheduling optimizations exploiting phases.

**Bars represent average results** over a range of assumptions about how much power the job is allocated and how much load imbalance is present.

**Experimental setup carefully emulates large-cluster load imbalance on a small cluster.**

**Results collected while running on Xeon hardware (not simulation).**
Presentation Outline

- GEO Project Overview
- GEO Architecture Overview
- Open Source Project Details (if time allows)
- Deep Dive: Application Feedback Interface
GEO Architecture Overview
GEO Hierarchical Architecture

GEO manages job to a power budget and globally coordinates frequency & power allocation decisions

Scaling challenge is addressed via tree-hierarchical design & hierarchical policy

- Each agent owns sub-problem: decide how to divide/balance power among children
- Power/perf telemetry is scalably aggregated so network traffic is minimal
- Tuning is globally optimized despite distributed tuning: achieved through Hierarchical-POMDP learning techniques

GEO tree runs in 1 reserved core per CN

- Leaf & non-leaf agents run in these cores
- Enables fast reaction times, deep analysis
- Overhead is negligible in manycore chips
- Designing for minimal memory footprint
Zoom-In on Leaf Agent

Power budgeting inside the processor:
- Spatio-Temporal Energy Scheduling (phase-adaptively allocate power among RAPL power domains)
Open Source Project Details
Team just completed first open source release on github

- Package Name: geopm (GEO power management)
- Release Goal: publish docs and interfaces for community review
- Non-Goal: feature-completeness
- Compatibility: Red Hat RHEL7 and SUSE SLES12 Linux distros
- Repository: view project and source code via http://geopm.github.io/geopm/
Release Notes

- Defined interfaces and architecture for integration in HPC SW stacks
- Nailed down our modular object-oriented design in C++11 (with C interfaces to external components / application)
- Developed solid autotools build system and gtest/gcov test infrastructure
- Delivered support for basic static power management functionality
  - E.g. Uniform Frequency Static mode
  - E.g. Hybrid Frequency Static mode (Pseudo Big Core / Little Core)
- No dynamic power management yet (still under construction)
  - No Auto-Tuner load balancing modes yet
Next Steps (Through Q1’16)

- **WIP on community adoption of GEO**
  - [DONE] Spin up collaborations with Argonne and LLNL
  - [WIP] Spin up collaborations with other national labs and universities
  - [WIP] Pursue community feedback on interfaces and documentation
  - [WIP] Joint research / publications with collaborators building on GEO

- **WIP on the runtime for dynamic power management**
  - [DONE] MPI communications between levels of GEO runtime hierarchy
  - [DONE] SLURM plug-in (initial development vehicle)
  - [DONE] Application feedback interface implementation
    - Recall: application markup is initially required for dynamic power mgmt modes
    - Long-term goal is for GEO to automatically infer the info without the API
  - [DONE] Extensibility in support for processor features
  - [WIP] Extensibility in decision algorithms
Deep Dive: Application Feedback Interface

Input $A_{OL}$ Output

$\beta$
Overview

- C interfaces provided in a lib that the app links against
  - They resemble typical profiler interfaces

- Consist of annotation functions for programmers to provide GEO info about app critical path and phases:
  - Indicate where bulk synchronizations occur (points where load imbalance results will result in degraded performance)
  - Indicate where phase changes occur in an MPI rank (i.e. phase entry and exit)
  - Indicate hints specifying whether phases will be compute-, memory-, or communication-intensive
  - Indicate how much progress each MPI rank has made toward completing the current phase (identify critical path)
Profiler Management / Reporting

```c
int geopm_prof_create(
    const char *name,
    size_t table_size,
    const char *sample_key,
    MPI_Comm comm,
    struct geopm_prof_c **prof);

int geopm_prof_destroy(
    struct geopm_prof_c *prof);

int geopm_prof_region(
    struct geopm_prof_c *prof,
    const char *region_name,
    long policy_hint,
    uint64_t *region_id);

int geopm_prof_print(
    struct geopm_prof_c *prof,
    const char *file_name,
    int depth);
```
Phase Markup / Bulk Sync Point

```c
int geopm_prof_enter(
    struct geopm_prof_c *prof,
    uint64_t region_id);

int geopm_prof_exit(
    struct geopm_prof_c *prof,
    uint64_t region_id);

int geopm_prof_outer_sync(
    struct geopm_prof_c *prof,
    uint64_t region_id);
```
Progress Reporting (1)

- Interfaces provide two options for reporting progress:
  - Special case (direct determination of critical path):
    - Assume: MPI+OpenMP w/ statically scheduled parallel regions
    - Assume: Total work for each individual thread is known
    - API computes rank’s progress as the min progress any thread made toward completing its total work (this is a %)
  - General case (estimation of critical path):
    - Assume: MPI+X
    - Assume: Total work is not known for each individual thread but the total work across all threads is known
    - API computes rank’s progress as sum of work completed on all threads / total work all threads will perform (this is a %)
Progress Reporting (2)

```c
int geopm_prof_progress(
    struct geopm_prof_c *prof,
    uint64_t region_id,
    double fraction);

int geopm_omp_sched_static_norm(
    int num_iter,
    int chunk_size,
    int num_thread,
    double *norm);

double geopm_progress_threaded_min(
    int num_thread,
    size_t stride,
    const uint32_t *progress,
    const double *norm);

double geopm_progress_threaded_sum(
    int num_thread,
    size_t stride,
    const uint32_t *progress,
    double norm);
```
Example of Application Markup (1)

max_threads = omp_get_max_threads();
posix_memalign((void **)&progress, cache_line_size,
    cache_line_size * max_threads);
memset(progress, 0, cache_line_size * max_threads);

norm = (double *)malloc(sizeof(double) * max_threads);
geopm_omp_sched_static_norm(num_iter, chunk_size,
    max_threads, norm);
geopm_prof_region(prof, "main-loop",
    GEOPM_POLICY_HINT_UNKNOWN, &region_id);
#pragma omp parallel default(shared) private(i, progress_ptr)
{
    progress_ptr = progress + stride * omp_get_thread_num();
    #pragma omp for schedule(static, chunk_size)
    for (i = 0; i < num_iter; ++i) {
        x += do_something(i);
        (*progress_ptr)++;
        if (omp_get_thread_num() == 0) {
            thread_progress = geopm_progress_threaded_min(
                omp_get_num_threads(), stride, progress, norm);
            geopm_prof_progress(prof, region_id, thread_progress);
        }
    }
}


Example of Application Markup (2)

```c
max_threads = omp_get_max_threads();
posix_memalign((void **)&progress, cache_line_size,
    cache_line_size * max_threads);
memset(progress, 0, cache_line_size * max_threads);
norm = 1.0 / num_iter;

geopm_prof_region(prof, "main-loop",
    GEOPM_POLICY_HINT_UNKNOWN, &region_id);
#pragma omp parallel default(shared) private(i, progress_ptr)
{
    progress_ptr = progress + stride * omp_get_thread_num();
    #pragma omp for schedule(static, chunk_size)
    for (i = 0; i < num_iter; ++i) {
        x += do_something(i);
        (*progress_ptr)++;
        if (omp_get_thread_num() == 0) {
            thread_progress = geopm_progress_threaded_sum(
                omp_get_num_threads(), stride, progress, norm);
            geopm_prof_progress(prof, region_id, thread_progress);
        }
    }
}
```
Coming Soon: Plug-In Interfaces

- Completion targeted for Q1’16 (hopefully early Q1)
- Platform plug-ins
  - Provides high-level abstraction of low-level processor interfaces for power & performance monitoring and control
    - E.g. control registers for RAPL, P-states, event counters, etc.
  - Simplifies porting to new Intel processors with new features (or processors from other vendors)
- Decider plug-ins
  - Enables researchers to extend GEO’s control algorithms
    - E.g. site-specific power management strategies
    - E.g. application-specific power management strategies
Backup Slides
Power Bounds

- Load imbalance is a big challenge
  - Apps tend to do bulk synchronizations
  - Performance is determined by last node to arrive at bulk synchronization point

- Power is becoming a scarce resource that must be managed carefully
  - Future systems are expected to be power-limited due to site limits
  - Processors are power-limited due to thermal design power limits

- Current strategies for managing power aggravate load imbalance
  - Uniform node power caps expose frequency variation from manufacturing variation
  - Uncoordinated Turbo/throttle decisions on nodes expose frequency variation
  - Results are far from optimal
Comparison Against Theoretical Bounds

- **Summary**
  - We achieved near-ideal benefits for most workloads with negligible losses vs. bounds
  - But, we note non-negligible losses of benefit for Integer Sort

- X-axis is a parameter for how much load imbalance we inject into the system
- Root-cause of benefit losses: some is initial search time, most is control error due to noise
- IS is considerably noisier than FFT and miniFE; working to improve handling of noise more

---

**Example of IS Losses w/ 90W Budget Config**

<table>
<thead>
<tr>
<th>% Delay</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>96.2%</td>
</tr>
<tr>
<td>10%</td>
<td>99.1%</td>
</tr>
<tr>
<td>20%</td>
<td>98.2%</td>
</tr>
<tr>
<td>30%</td>
<td>97.8%</td>
</tr>
<tr>
<td>40%</td>
<td>96.4%</td>
</tr>
</tbody>
</table>
GEO Advanced Power Balancing Modes

Can configure objective function for how GEO will dynamically mitigate imbalance
- a) Equalize processor frequency
- b) Equalize node’s app progress (steer power to critical path)