CNeRF + The Sim Theory Scheduler: A Case Study

Goal

In December 2024, Simulation Theory needed a good candidate project to demonstrate how an application using an optimized framework can benefit from a minimal code impact, low-effort integration with Sim Theory's proprietary Scheduler technology.

The requirements were:

- ➔ A CPU-intensive computational load
- ➔ Use of common frameworks
- ➔ Open source
- ➔ Implemented in C/C++

CNeRF[1] was chosen because it meets the requirements in the following ways:

- ➔ NeRF[2] methods are computationally intensive
- ➔ CNeRF[3] is implemented using PyTorch (a commonly used compute framework)
- ➔ PyTorch uses OpenMP internally for CPU parallelization
- ➔ CNeRF is open source with an MIT license
- ➔ CNeRF is implemented in C/C++

[1] https://github.com/rafaelanderka/cNeRF
[2] A neural radiance field (NeRF) is a method for reconstructing a three-dimensional representation of a scene from two-dimensional images based on deep learning. It is computationally expensive because the NeRF algorithm represents a scene as a radiance field parametrized by a deep neural network (DNN). The network predicts a volume density and view-dependent emitted radiance given the spatial location (x, y, z) and viewing direction in Euler angles (θ, Φ) of the camera. By sampling many points along camera rays, traditional volume rendering techniques can produce an image.
[3] CNeRF provides a minimal implementation of Neural Radiance Fields (NERF) which is a method for synthesizing novel views of complex scenes using neural inverse modelling. The code is written in C++ and utilizes LibTorch for automatic differentiation.

Hypothesis

Simulation Theory can improve CNeRF's CPU occupancy by 30% or more if Sim Theory's Scheduler is integrated at a top-level domain in the application. This work should not disrupt or modify the fork and join nature of OpenMP within PyTorch.

Out of the box, running on the CPU, CNeRF was achieving an effective CPU occupancy of 39.5% using about 25 of the 64 available logical CPU cores. This performance is due to PyTorch, its use of OpenMP, and the parallelization OpenMP provides.

Integration

Simulation Theory approached integration with CNeRF in two ways:

- ➔ **Subdivide the work into equally-sized blocks.** The Scheduler was initialized with *N* threads. All work was then broken up inside the Scheduler into *N* evenly-sized blocks and processed one block per thread. The result of this is a small number of large tasks processing simultaneously on *N* threads.
- ➔ **Place all work in a work buffer using an atomic incrementor.** The Scheduler was initialized with *N* threads. An atomic incrementor was used to gate access by the *N* threads to the data buffer. This is the most efficient use of the Scheduler and results in fine-grained tasks processing simultaneously on *N* threads.

Results

Within a 4 hour development lifecycle, a member of the Simulation Theory team was able to integrate our Scheduler technology into the CNeRF project using two different work distribution models.

Data Processing Time

Using the same data set as the baseline test along with the new fine-grained work distribution implementation, the Simulation Theory Scheduler was able to **reduce** the total processing time from approximately **11 hours** to **7.5 hours**, a reduction of about **31.8%**. We expect that we can demonstrate a more significant reduction, however, the testing system was disk IO bound during testing.

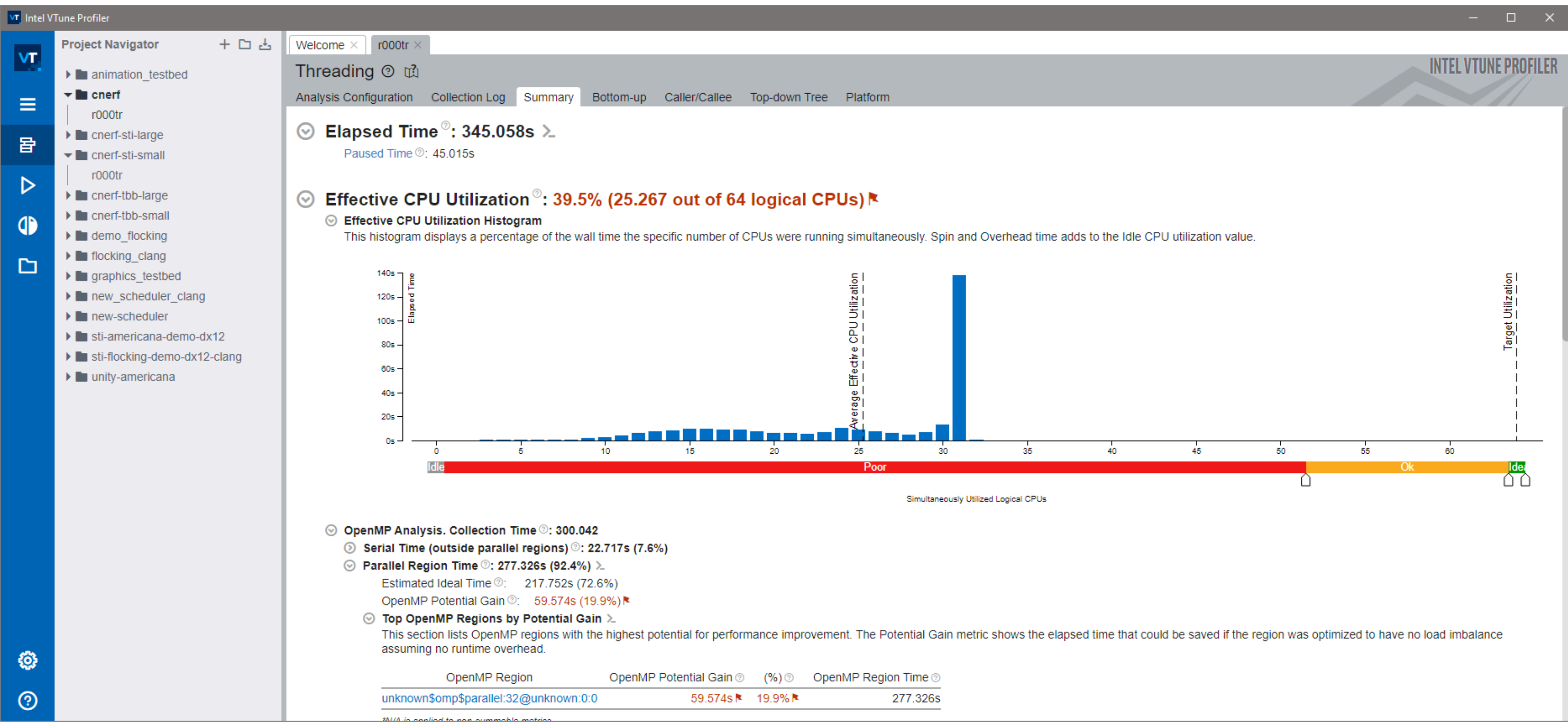
CPU Occupancy

The overall CPU occupancy was **39.5%** on 25.267 out of 64 logical CPU cores in the baseline implementation of CNeRF.

Using the large-block work distribution model of the Simulation Theory Scheduler implementation, CPU occupancy **improved** to **57.4%** on 36.735 out of 64 logical CPU cores, an **improvement of 31.2%**.

Using the fine-grained work distribution model of the Simulation Theory Scheduler implementation, CPU occupancy **improved** to **65.2%** on 41.743 out of 64 logical CPU cores, an **improvement of 39.47%**.

CNeRF Original Performance Graph[4]



Elapsed Time: 345.058s

Paused Time: 45.015s

Effective CPU Utilization: 39.5% (25.267 out of 64 logical CPUs)

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.

OpenMP Analysis. Collection Time: 300.042

Serial Time (outside parallel regions): 22.717s (7.6%)

Parallel Region Time: 277.326s (92.4%)

Estimated Ideal Time: 217.752s (72.6%)

OpenMP Potential Gain: 59.574s (19.9%)

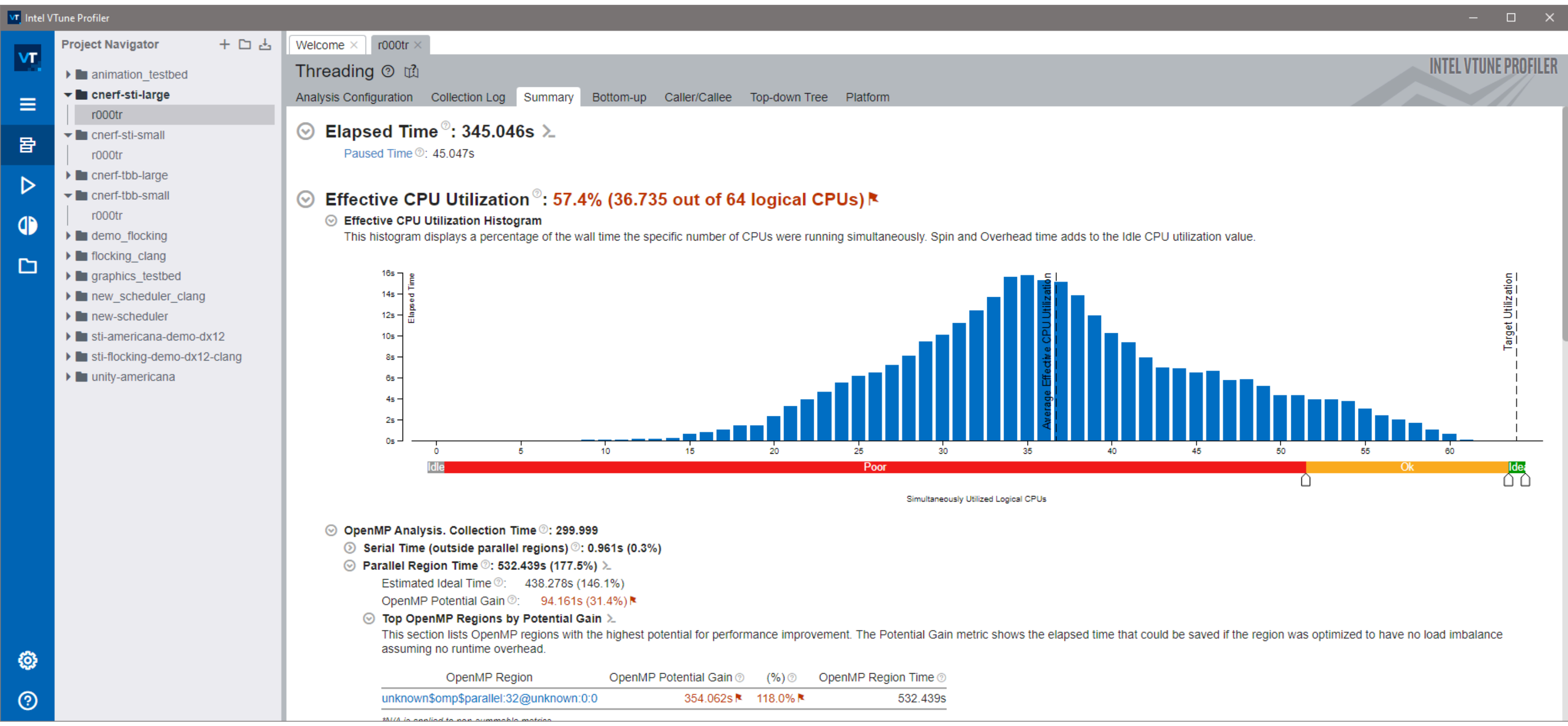
Top OpenMP Regions by Potential Gain

This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metric shows the elapsed time that could be saved if the region was optimized to have no load imbalance assuming no runtime overhead.

OpenMP Region	OpenMP Potential Gain	(%)	OpenMP Region Time
unknown$omp$parallel:32@unknown:0:0	59.574s	19.9%	277.326s

[4] Simulation Theory used Intel V-Tune to capture these and the following performance metrics from the testing.

CNeRF Simulation Theory Large, Equally-Sized Tasks Performance Graph



Threading

Elapsed Time: 345.046s

Paused Time: 45.047s

Effective CPU Utilization: 57.4% (36.735 out of 64 logical CPUs)

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.

OpenMP Analysis. Collection Time: 299.999

Serial Time (outside parallel regions): 0.961s (0.3%)

Parallel Region Time: 532.439s (177.5%)

Estimated Ideal Time: 438.278s (146.1%)

OpenMP Potential Gain: 94.161s (31.4%)

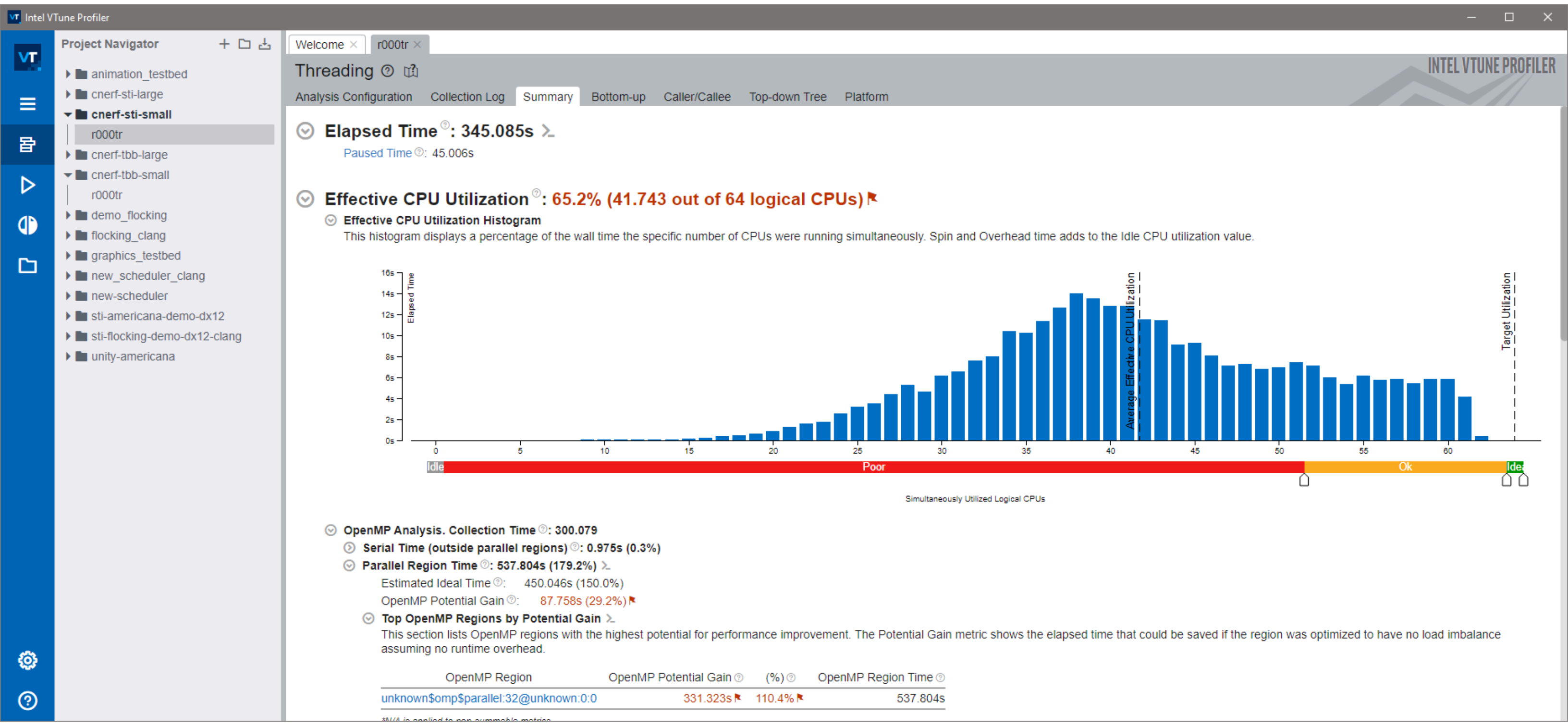
Top OpenMP Regions by Potential Gain

This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metric shows the elapsed time that could be saved if the region was optimized to have no load imbalance assuming no runtime overhead.

OpenMP Region	OpenMP Potential Gain	(%)	OpenMP Region Time
unknown$omp$parallel:32@unknown:0:0	354.062s	118.0%	532.439s

*N/A is applied to non-summable metrics

CNeRF Simulation Theory Small Tasks Performance Graph



Elapsed Time: 345.085s

Paused Time: 45.006s

Effective CPU Utilization: 65.2% (41.743 out of 64 logical CPUs)

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.

OpenMP Analysis. Collection Time: 300.079

Serial Time (outside parallel regions): 0.975s (0.3%)

Parallel Region Time: 537.804s (179.2%)

Estimated Ideal Time: 450.046s (150.0%)

OpenMP Potential Gain: 87.758s (29.2%)

Top OpenMP Regions by Potential Gain

This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metric shows the elapsed time that could be saved if the region was optimized to have no load imbalance assuming no runtime overhead.

OpenMP Region	OpenMP Potential Gain	(%)	OpenMP Region Time
unknown$omp$parallel:32@unknown:0:0	331.323s	110.4%	537.804s

Author: Randy Culley, CTO
sales@simtheoryinc.com
simtheoryinc.com