High-Performance Computing Report: Kernel Density Estimation Optimization using OpenMP

AUTHORS

Pedro Henrique Nunes Souza

AFFILIATIONS

Instituto Politécnico de Beja

Large datasets can pose significant computational challenges. This poster shows how Kernel Density Estimation (KDE) can be optimized with OpenMP on multicore CPUs and GPUs. The aim is to cut execution time, boost scalability, and make KDE practical for various high-performance tasks, from machine learning to financial modeling.

Instituto Politécnico de Beja — Escola Superior de Tecnologia e Gestão

01. Introduction

Kernel Density Estimation (KDE) is a non-parametric method for estimating the probability density function of a random variable. Despite its broad applicability in fields such as machine learning, image processing, and finance, KDE has a notable problem: its computational complexity can be as high as $O(n^2)$. As the size of datasets grows, this quickly becomes a bottleneck. This poster presents how parallel computing, using OpenMP directives on both multicore CPUs and GPUs, can significantly reduce KDE's execution time and make it more practical for real-world, large-scale applications.

02. Objective

The main objective is to demonstrate the effectiveness of parallelization strategies for speeding up KDE computations:

1. **Reduce Execution Time:** Show how distributing workload across CPU cores or offloading it to a GPU can cut down processing time.
2. **Enhance Scalability:** Test different dataset sizes to evaluate whether the proposed parallel solutions can handle larger volumes more efficiently.
3. **Compare Approaches:** Examine both CPU-based (multicore) and GPU-based offloading to identify which method performs better under various conditions.

03. Methodology

Baseline (Sequential):
Runs basic KDE with a nested loop on a single CPU core as a speedup benchmark.
Multicore CPU (OpenMP):
Uses \texttt{#pragma omp parallel for}, \texttt{collapse(2)}, and \texttt{reduction} to split the nested loop among multiple CPU cores.
GPU Offloading (OpenMP):
Employs \texttt{#pragma omp target} to move data and computation to the GPU, leveraging directives like \texttt{teams distribute parallel for} and explicit \texttt{map} clauses.
Performance Metrics:

- Time: Measured via *omp_get_wtime()*.
- *Speedup (S) = T(sequential) / T(parallel).*
- Scalability: Evaluated at dataset sizes of 64, 128, 256, and 512.

$$S = \frac{T_{sequencial}}{T_{parallel}}$$

04. Results/Findings

During the experiments, each approach (Sequential, Multicore CPU, GPU Offloading) was timed. The multicore CPU version utilized multiple threads, while the GPU version attempted to harness the massive parallelism of graphics cards.

Key Observations:

- The multicore CPU approach achieved a speedup close to 3×, illustrating that distributing computations across multiple CPU cores can significantly reduce the runtime.
- The GPU offloading solution showed limited benefits for smaller datasets, primarily due to overhead from data transfers between CPU and GPU. For larger datasets, there is potential for greater gains if memory transfers are optimized.

```
Zellij (stellar-drum)  Tab #1
 tei26699@rolf: ~/hpc/build
tei26699@rolf:~/hpc/build$ cmake ../repo
-- The C compiler identification is GNU 12.3.0
-- The CXX compiler identification is GNU 12.3.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenMP_C: -fopenmp (found version "4.5")
-- Found OpenMP_CXX: -fopenmp (found version "4.5")
-- Found OpenMP: TRUE (found version "4.5")
-- Configuring done
-- Generating done
-- Build files have been written to: /home/alunos/tei/2024/tei26699
tei26699@rolf:~/hpc/build$ make
[ 16%] Building CXX object CMakeFiles/kde_sequential.dir/kde_sequ
[ 33%] Linking CXX executable kde_sequential
[ 33%] Built target kde_sequential
[ 50%] Building CXX object CMakeFiles/kde_parallel.dir/kde_parall
[ 66%] Linking CXX executable kde_parallel
[ 66%] Built target kde_parallel
[ 83%] Building CXX object CMakeFiles/kde_gpu.dir/kde_gpu.cpp.o
[100%] Linking CXX executable kde_gpu
[100%] Built target kde_gpu
tei26699@rolf:~/hpc/build$ ./kde_sequential
./kde_parallel
./kde_gpu
Tempo (Sequencial): 0.0446071 segundos
Tempo (Multicore OpenMP): 0.0160048 segundos
Tempo (GPU OpenMP Offloading): 0.327597 segundos
tei26699@rolf:~/hpc/build$
```

05. Analysis

- **CPU Parallelization:** The near 3× speedup confirms that OpenMP's thread-level parallelism can effectively divide the double loop among CPU cores.
- **GPU Offloading:** While smaller datasets may not express substantial gains due to transfer overhead, GPUs can excel with larger workloads.

Size	Sequential (s)	Multicore (s)	GPU (s)	Speedup (Multicore)	Speedup (GPU)
64	0.0448066	0.0153747	0.324886	2.9143	0.1379
128	0.0448795	0.0150333	0.297027	2.9853	0.1510
256	0.04557	0.0146112	0.297871	3.1188	0.1529
512	0.0445145	0.0141523	0.292162	3.1453	0.1523

06. Conclusion

Parallelization with OpenMP effectively accelerates KDE by distributing computations across multiple CPU cores or offloading them to the GPU. Although high data-transfer costs can affect GPU performance on smaller datasets, careful memory management and optimization strategies can make GPU offloading a powerful option for large-scale KDE workloads.

07. Bibliografia

- Weglarczyk, S. (2018) Kernel Density Estimation: Methods and Applications. (Discusses theoretical and applied aspects of KDE.)
- Langrené, N. et al. (2017) Fast KDE Techniques in High-Performance Computing. (Explores parallel strategies for accelerating KDE.)
- Wikipedia "Kernel Density Estimation," \url{https://en.wikipedia.org/wiki/Kernel_density_estimation}(Provides an overview of KDE theory and examples.)