Comparing the Performance of C++ Parallel Algorithms With their Implementations in HPX

Jean B. Beldunord\textsuperscript{1}, Parsa Amini\textsuperscript{2,3}, Dr. Hartmut Kaiser\textsuperscript{2,3}

\textsuperscript{1}Florida Memorial University, \textsuperscript{2}Louisiana State University, \textsuperscript{3}Stellar Group

Introduction

Traditionally, computer programming was conducted in a sequential manner, similar to the operation of our brain. Computers use two main aspects in order to operate. Hardware where it’s highly diverse and parallel, and software which is mainly sequential. Consequently a small portion of hardware can be utilized, which causes performance issues. Parallel computing allows us to execute tasks concurrently. Two types of software that are currently developing parallel algorithms are: HPX, which is a general purpose C++ runtime system for parallel and distributed applications of any scale, and C++ 17. I will be evaluating their algorithms.

Execution policy parameter

1. Sequence policy – An execution policy that a parallel algorithms’ execution may not be parallelized.
2. Parallel policy – Showcase that a parallel algorithm may be parallelized
3. Parallel unsequenced policy - An execution policy showcasing that a parallel algorithm may be parallelized or vectored.

Algorithms used

1. Reduce Parallel: The reduce algorithm returns the result of the generalized sum over the elements given by the input range [first, last]
2. Find Parallel: The find algorithm is used to locate different instances.

Procedure

- Utilize a benchmark with the two algorithms mentioned on a vector.
- Ran across both HPX and C++ application on Visual studio. Using a Pc with twelve cores.
- Used appropriate sample size, where difference are significant with a 95% confidence interval
- Graph the results of selected algorithms.

Data

Reduce Par Performance Test

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{reduce_par_performance_test.png}
\caption{Reduce Par Performance Test}
\end{figure}

Find Par Performance Test

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{find_par_performance_test.png}
\caption{Find Par Performance Test}
\end{figure}

Conclusion

From the collected results, I observed that HPX utilized more of the hardware when running on more cores. While the C++ 17 uses a different number of cores it takes about the same amount of time for each process. The reason for that is because HPX took advantage of hyperthreading, where you can change the amount of threads used. It almost reaches perfect scaling as the number of cores increases. Overall performance wise, the minimum time used by both is closely related.

References

- Filipek, Bartlomiej. “Examples of Parallel Algorithms From C++ 17,” Bartek’s Coding Blog, 25 June 2018

Acknowledgement

- Work supported by the National Science Foundation (NSF) award #ACI-1560410.
- A special thanks To Adrian Serio & the rest of the Stellar Group team.
- Thank you to the LSU REU Team for such a unforgettable summer experience. I will forever be grateful.