The end of Dennard scaling and the rising popularity of data analytics and machine learning have resulted in a rapid increase in the adoption of heterogeneous parallelism. Graphics Processing Units (GPU) and Field Programmable Gate Arrays (FPGA) have evolved from being used as accelerators in niche application areas to being an integral part of almost all cloud computing platforms. This has led to a surge in interest in the design of database systems that can exploit such XPU architectures instead of the CPU. However, a major factor that has limited the wide-spread adoption of XPU in database systems has been the lack of portable parallelism. Historically, general-purpose, standardized programming languages like C++ were developed before the evolution of heterogeneous parallel computing. Thus, there was no common software stack for programming XPU, and developers had to use vendor-specific programming platforms and APIs. This problem was particularly acute in High-Performance Computing (HPC), where limited compiler support for a particular combination of architecture and programming model forced HPC developers to maintain multiple versions of codes in each programming model. The need for a portable programming model that enables just one version of the source code to work across diverse architectures led to the development of OpenCL, a cross-industry framework initiative that provided a ``C''-like language for writing compute-intensive kernel that can be offloaded onto any supported XPU using a runtime API.

Over the past few years, HPC applications have evolved to adopt more general-purpose programming languages like C++ instead of C and FORTRAN, and HPC installations expanded to adopt XPU from more vendors. This change exposed several limitations of low-level frameworks like OpenCL. First, the low-level nature of OpenCL was meant to directly expose data parallelism in underlying hardware while leaving everything else from data movement to kernel dispatch to developers leading to boilerplate code verbosity. Second, programs written in OpenCL are not single-source in nature as kernel code needs to be separated from host code, represented as strings and separately managed complicating software development. These challenges led to the development of custom HPC frameworks like RAJA~\cite{raja} and Kokkos~\cite{kokkos} that bridge the gap by providing C++ abstraction layers for portable parallel execution. Thus, in turn, spurred the development of SYCL, an open, industry-standard, single-source, modern C++ parallel programming model from Khronos group (who also maintain OpenCL).

This thesis will investigate the utility of SYCL in the development of performance-portable data management engines for several data intensive domains. First, we focus on relational database engines. We will start from a state-of-the-art, data-parallel hash join that has been developed in CUDA and optimized for execution on NVIDIA GPU. We will port the join to Data-Parallel C++ (DPC++)--an open-source implementation of SYCL--using the oneAPI toolkit. We will then execute it on \supertextreg{Intel} multicore CPU, integrated \supertextreg{Intel} GEN9 GPU, \supertextreg{Intel} \supertextreg{Iris} Xe Max DG1 discrete GPU, and NVIDIA RTX 2080 Ti discrete GPU, without changing the data-parallel kernel code to demonstrate cross-architecture, cross-vendor portability of DPC++. Using models that provide theoretical upper bounds for CPU performance, and using the state-of-the-art, hand-optimized CUDA join, we will investigate the performance of the SYCL-join to understand the effectiveness of SYCL in parallelizing on CPU and GPU. Following this, we will investigate the performance-portable acceleration of dynamic programming – a key algorithm

used in several applications, from query optimization in databases to sequence alignment in genomics. We will investigate the newly released DPX extensions in CUDA and investigate their utility in accelerating DP formulations for a few applications.