Abstract

Scientific applications that operate on large data sets require huge amount of computation power and memory. These applications are typically run on High Performance Computing (HPC) systems that consist of multiple compute nodes, connected over an network interconnect such as InfiniBand. Each compute node has its own memory and does not share the address space with other nodes. A significant amount of work has been done in past two decades on parallelizing for distributed-memory architectures. A majority of this work was done in developing compiler technologies such as high performance Fortran (HPF) and partitioned global address space (PGAS). However, several steps involved in achieving good performance remained manual. Hence, the approach currently used to obtain the best performance is to rely on highly tuned libraries such as ScaLAPACK. The objective of this work is to improve automatic compiler and runtime support for distributed-memory clusters for regular programs. Regular programs typically use arrays as their main data structure and array accesses are affine functions of outer loop indices and program parameters. A lot of scientific applications such as linear-algebra kernels, stencils, partial differential equation solvers, data-mining applications and dynamic programming codes fall in this category.

In this work, we propose techniques for finding computation mapping and data allocation when compiling regular programs for distributed-memory clusters. Techniques for
transformation and detection of parallelism, relying on the polyhedral framework already exist. We propose automatic techniques to determine computation placements for identified parallelism and allocation of data. We model the problem of finding good computation placement as a graph partitioning problem with the constraints to minimize both communication volume and load imbalance for entire program. We show that our approach for computation mapping is more effective than those that can be developed using vendor-supplied libraries. Our approach for data allocation is driven by tiling of data spaces along with a compiler assisted runtime scheme to allocate and deallocate tiles on-demand and reuse them. Experimental results on some sequences of BLAS calls demonstrate a mean speedup of $1.82 \times$ over versions written with ScaLAPACK. Besides enabling weak scaling for distributed memory, data tiling also improves locality for shared-memory parallelization. Experimental results on a 32-core shared-memory SMP system shows a mean speedup of $2.67 \times$ over code that is not data tiled.