Practical Experience with the AllScale Programming Model

Herbert Jordan[a], Thomas Heller[b], Peter Zangerl[a], Philipp Gschwandtner[a], Peter Thoman[a], and Thomas Fahringer[a]

[a] *University of Innsbruck*
[b] *Friedrich-Alexander University of Erlangen-Nuremberg*

As part of the EU FETHPC H2020 AllScale project, we have researched a novel programming model for HPC applications based on nested recursive parallelism. We provisioned an effective and efficient, purely C++ based, target platform oblivious, shared-memory like programming interface for large-scale HPC applications. Through this interface, the application specific code remains isolated from the implementations of parallel data structures and operators, data exchange mechanisms, synchronization primitives, and resource management services like load balancing, monitoring, and tuning components. The AllScale architecture isolates each of those responsibilities, facilitating the orthogonal development of each, while, at the same time, ensuring a seamless, automated integration as part of the application compilation and linking process. Its simpler, shared-memory like programming API facilitates the development of high-performance distributed memory applications, exhibiting comparable scalability to state-of-the-art MPI implementations, without the explicit management of work and data distributions.

The AllScale architecture comprises three layers: a C++ API layer, a source-to-source compiler, and a distributed memory runtime system. The API layer provides a set of parallel primitives (e.g. generic parallel for loops, reductions, and stencil operators) as well as a range of data structures (e.g. generic multi-dimensional grids, adaptive grids, trees, and graph data structures). All parallel operators are reduced within the API to a common, higher-order parallel operator, the *prec* operator [1]. Applications built on top of the API can be developed and compiled using conventional C++ infrastructure, producing shared-memory parallel applications. However, when compiled using the AllScale source-to-source compiler, parallel regions are identified, analyzed, and, if applicable, converted into distributed memory code. High-level static program analyses are utilized to identify data requirements of parallel regions implicitly, without the need of the user to specify those explicit. The obtained data requirements are used by the compiler to synthesize code fitting the AllScale Runtime System application model [2]. The resulting programs can be processed by the runtime system on a distributed memory system, handling the data and work distribution automatically.

The AllScale infrastructure has been deployed on LEO-2 (UIBK), VSC-3 (Vienna), Beskow (KTH), Meggie (FAU), SuperMUC (Germany), and Cori (NERSC). Furthermore, three real-world physical simulation codes have been ported to the infrastructure. In all three cases, the AllScale programming model significantly reduced the code complexity of parallel regions. Corresponding lines of code could be reduced by 58%-73%. Scalability experiments on up to 5120 cores and 256 nodes demonstrated comparable performance to MPI in a static, evenly load-balanced input use case. For use cases exhibiting load-imbalance the integrated application-independent inter-node load balancing support results in up to 10% performance increase for realistic inputs to our simulation codes.

References

[1] Jordan, H., Thoman, P., Zangerl, P., Heller, T., and Fahringer, T, A Context-Aware Primitive for Nested Recursive Parallelism, European Conference on Parallel Processing, 149 (2016)

[2] Jordan, H., Heller, T., Gschwandtner, P., Zangerl, P., Thoman, P., Fey, D., and Fahringer, T, The AllScale Runtime Application Model, International Conference on Cluster Computing, 445 (2018)