Using Advanced Vector Extensions AVX-512 for MPI Reduction

Dong Zhong, George Bosilca, Qinglei Cao

Headline

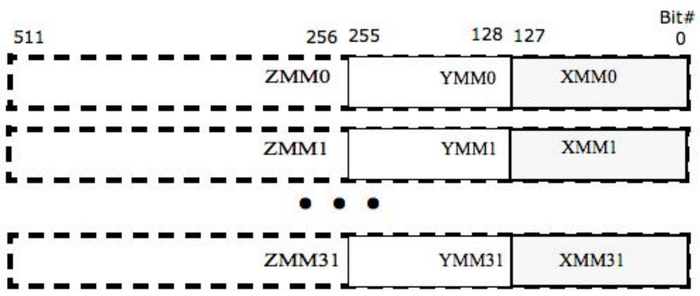
As the scale of high-performance computing (HPC) systems continues to grow, researchers are devoted themselves to implore increasing levels of parallelism to achieve optimal performance. Novel processors support wide vector extensions, vectorization becomes much more important to exploit the potential peak performance of target architecture.

New processor architectures, such as, Intel introduced 512-bit extensions (AVX-512) to the 256-bit Advanced Vector Extensions (SIMD) instructions for x86 instruction set architecture (ISA). ARM's new Armv8-A architecture, introduce Scalable Vector Extension (SVE)- an optional separate architectural extension with a new set of A64 instruction encodings, which enables even greater parallelisms.

In this paper, we propose new strategies by utilizing those instructions to improve the performance of MPI reduction operations. With these optimizations, we not only provide a higher-parallelism for a single node, but also achieve a more efficient communication scheme of message exchanging. The resulting efforts have been implemented in the context of OMPI. The evaluation of the resulting software stack under different scenarios with Skylake processor demonstrates that the solution is at the same time generic and efficient.

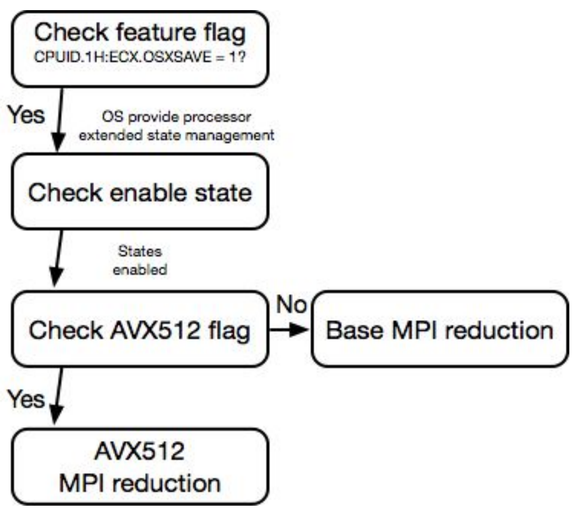
AVX-512 instructions

AVX-512 instructions support 512-bit wide SIMD registers (ZMM0-ZMM31). The lower 256-bits of the ZMM registers are aliased to the respective 256-bit YMM registers and the lower 128-bit are aliased to the respective 128-bit XMM registers.



AVX-512 support check

Automatically detect the hardware information to enable AVX-512 reduction feature or fallback to basic module if it is not supported



AVX-512 reduction

Reduction algorithm with avx512,256 and duff device

Algorithm 1 AVX based reduction algorithm

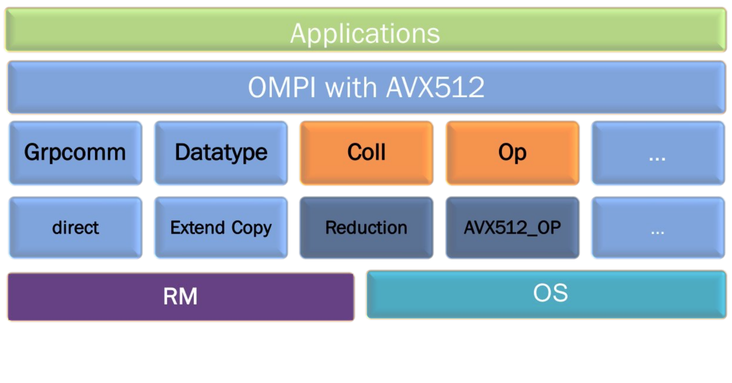
types_per_step ▷ Number of elements in vector
left_over ▷ Number of elements waiting for reduction
count ▷ Total number of elements for reduction operation
in_buf ▷ Input buffer for reduction operation
inout_buf ▷ Input and output buffer for reduction operation

```
1:  procedure REDUCTIONOP( in_buf, inout_buf, count )
2:      types_per_step = vector_length(512) / (8 × sizeof_type)
3:      for k ← types_per_step to count do
4:          _mm512_loadu_si512 from in_buf
5:          _mm512_loadu_si512 from inout_buf
6:          _mm512_reduction_op
7:          _mm512_storeu_si512 to inout_buf
8:      if ( left_over ≠ 0 ) then
9:          Update types_per_step >>= 1
10:         if ( types_per_step ≤ left_over) then
11:             _mm256_loadu_si256 from in_buf
12:             _mm256_loadu_si256 from inout_buf
13:             _mm256_reduction_op
14:             _mm256_storeu_si256 to inout_buf
15:     if (left_over ≠ 0 ) then
16:         Duff device
```

Design and implementation in OMPI

We implemented AVX512 this work in a set of components in OMPI which is based on a Modular Component Architecture that permits easily extending or substituting the core subsystem with new features.

As shown below, we added our AVX512 optimization work in a components to OMPI architecture that implements all MPI reduction operations with AVX512 vector reduction instructions; to be noted, this component can be extend out the scope of local reduction to general mathematics and logic operations.

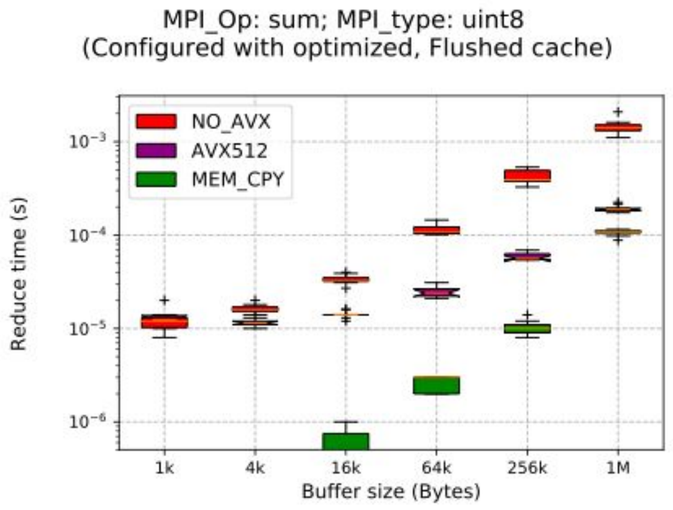


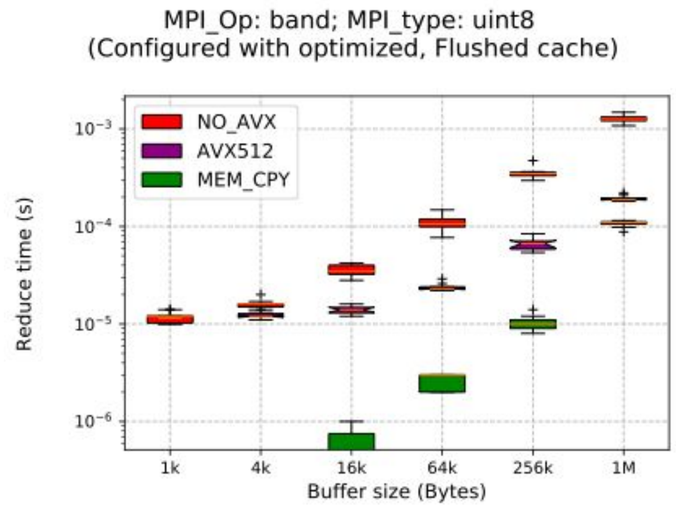
Experimental evaluation

Experimented on a local cluster which is a Intel(R) Xeon(R) Gold 6254 based server running at 3.10 GHz.Our work is based upon OMPI master branch, revision \#75a539. Each experiment is repeated 30 times and we present the average. For all experiment we use a single node with one process, because our optimization aims to improve the performance of local reduction operation.

We compare arithmetic SUM and logical BAND. For the experiments we flushed cache to ensure we are not reusing cache for fair comparison. Results demonstrate that with AVX512-enabled operation it is ~10x faster than element-wise operation. We also compare MPI operation together with memcpy which indicates the peak memory bandwidth. MPI Op=(2 loads + 1 store + computation), Memcpy = (1 load + 1 store).

It shows even with computation included AVX512 reduction operation achieves a similar level of memory bandwidth as memcpy.





ICL INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY OF TENNESSEE KNOXVILLE

SPONSORED BY
U.S. DEPARTMENT OF ENERGY | Office of Science
ECP EXASCALE COMPUTING PROJECT
NNSA National Nuclear Security Administration