Towards benchmarking the asynchronous progress of non-blocking MPI point-to-point and collective operations

Alexey V. Medvedev

Institute of mechanics, Lomonosov Moscow State University, a.medvedev@imec.msu.ru

Introduction

The non-blocking MPI communications, both point-to-point and MPI-3 non-blocking collectives are important tools which allow the design of MPI-based parallel algorithms that avoid rigid, fully blocking communication patterns. The use of non-blocking MPI operations also makes it possible to introduce algorithm designs which imply calculation/communication overlapping to hide the cross-rank communication latency. The problem in the latter case is that MPI standard does not require from MPI library implementers to make non-blocking communication operations partially or fully asynchronous. This means that for a good level of calculation/communication overlapping some effort from both middleware implementers and parallel code authors may be required.

To estimate which level of asynchronous message passing progress at the same time with intensive calculations can be expected on a particular machine using a particular communication middleware setup, the general purpose benchmarking tools can be used. The popular MPI micro-benchmarks OSU [1] and IMB [2] include only a small subset of scenarios that may help to estimate which calculation/communication overlapping level might be available for some practical applications. The authors faced at least two cases in their work when existing benchmarking tools do not give a full picture: i) in sparse linear algebra applications, there are several research results [4] offering the non-blocking MPI_Iallreduce call usage in Krylov subspace iterative methods to hide the latency of dot product calculations; ii) point-to-point communications with neighbors are also widely used (in their non-blocking forms namely) in sparse linear algebra applications due to good potential of communication latency hiding. Before the new MPI-based code development, it is nice to estimate which overlapping can be expected in both cases, and how the overlapping depends on a particular middleware/hardware and how it changes at scale: these estimations help a lot at the algorithm choice stage. This necessity is not limited to sparse linear algebra problems.

The goal of this work is to offer the basic and practical design of some new benchmarks which may meet the demand of parallel algorithm developers challenged with the problem of hiding the communication latency and to offer the calculation/communication overlapping efficiency estimation method.

The group of benchmark codes described in this paper is a newly developed extension to the open-source Intel MPI Benchmark 2019 suite, made by authors as a separate project [3]. The name for the benchmark group – IMB-ASYNC – is chosen to match the naming conventions of other IMB benchmark groups. The difference between the benchmarks in question and older parts of IMB suite is that all of their non-blocking variants include tunable computational load simulation procedure. The time for this procedure to produce CPU load can be set up from the command line with ~1 microsecond precision. This makes it possible to estimate calculation/communication overlap carefully for every scenario.

Benchmark structure

The IMB-ASYNC code offers:

i) point-to-point benchmarks estimating the pair-wise communications between regular pairs of ranks from the MPI_COMM_WORLD communicator. The ping-pong style communications in each pair are

implemented in two ways: "sync_pt2pt" is a sequence of blocking MPI_Send and MPI_Recv calls (reversed in its order in one of the ranks of a pair) and "async_pt2pt" is a sequence of non-blocking MPI_Isend, MPI_Irecv, followed by optional block, simulating high computational load, with MPI_Waitall call finishing the benchmark iteration. The design of blocking kind of this benchmark is made similar to a well-known "PingPong" benchmark of IMB-MPI1 suite.

ii) point-to-point benchmark estimating the neighborhood communications between a given rank and two or more neighbor ranks. It also has two implementations: blocking ("sync_exch") and non-blocking ("async_exch") flavors, and is equipped with the same optional computational load simulation procedure for a non-blocking kind.

iii) collective benchmark ("sync_allreduce", "async_allreduce") estimating blocking and non-blocking kind of one of most popular collective operations. Non-blocking kind is also equipped with the same optional computational load simulation procedure.

Asynchronous progress implementation details

The asynchronous progress of MPI operations may be implemented in different ways. It is uncommon for up-to-date MPI libraries to have "transparent", hardware-supported asynchronous progress for all possible non-blocking operations and message sizes. However, there are techniques at the application level which may "stimulate" or "accelerate" the asynchronous progress in some particular cases. The specialized "progression threads" usage (like MPICH_ASYNC_PROGRESS-related technique for MPICH-based implementation) is often discussed in literature. Also, the popular way is: to do "manual progression" via periodical calls to MPI_Test or MPI_Probe functions during the progress of computations. To estimate the asynchronous message passing progress which may be achieved with the latter approach, the computational load simulation code of the IMB-ASYNC benchmarks includes the turnable and tunable MPI_Test calls for all the MPI requests appeared in a benchmark.

Efficiency estimation method and computational experiments

The benchmark code was tested on Lomonosov-2 supercomputer of the Lomonosov Moscow State University. A short selection of the benchmarking results is given in Table 1 and Table 2.

The full subscription of 2, 16 and 64 computing nodes (14 cores and 14 ranks per node with per-core affinity) was used to estimate the asynchronous progress expectations with Intel MPI 2017.1 library. The baseline is the time taken by a blocking version of each benchmark (see "Pure communication time, blocking MPI" table rows). The main goal of the estimation is to understand the time benefit one may have by hiding communications behind the calculations for each message. For this purpose, the simulative calculation procedure in the non-blocking versions of benchmarks is turned on. Amount of computing cycles is chosen manually so that communication time was 1.5-2 times less than the computation time (see "Calculation time" table rows). This way to choose calculation cycle length is chosen because it corresponds to the majority of use cases: normally calculation-intensive algorithms have enough CPU workload to hide calculation latency, and the situations when calculation time is comparable with communication time seems to be a cormer case of extreme scaling attempts.

The communication overhead which appears in this running mode (see "Communication overhead..." table rows) is then compared to the baseline time of blocking communications to find out the Communication Hiding Efficiency:

$$E = 100\% - (T_{over} / T_{sync}) * 100\%$$

where T_{over} is the communication overhead time in calculation/communication mode, T_{sync} is the baseline.

The Efficiency of 100% means that thanks to non-blocking communication form usage, all the communications are successfully hidden by computations. This is the best possible case. The worst case is

when Efficiency values become negative: it means that combining the non-blocking communications with calculations is even slower than the sequential combination of calculations and blocking communications.

Please note that this Efficiency estimation method is different from the one used in IMB-NBC suite [5]. IMB-NBC compares the communication overhead time in calculation/communication mode with the pure latency of anynchronous communication call without calculations. This way of estimating the efficiency is not quite practical, because the choice is normally made between two modes: synchronous communications serialized with calculations or non-blocking communications with the goal to overlap them with communications. Moreover, in IMB-NBC methodology, it is difficult to estimate correctly which penalty the progress thread introduces since both estimation of baseline time and the benchmark of interest are made during the same execution session, and possible penalty influences both figures. The IMB-NBC suite always use the calculation time equal to baseline communication time. This doesn't give the possibility to adjust the calculation/communication overlap case according to the practical interest of the algorithm designer. Also, IMB-NBC doesn't try to estimate possible efficiency loss due to calculations slowdown when progress thread technique is used to support asynchronous progress.

Two kinds of progress-support techniques were estimated: the "Manual progress" table rows show the results of enabling manual progression via periodical MPI_Test calls in the benchmark code; the "MPICH_ASYNC_PROGRESS" rows show what the internal progress-thread based technique of Intel MPI 2017.1 library may offer instead. The rows "Efficiency of overlapping ..." give the Efficiency value estimation for each of the two progress-support techniques tested (E_1 and E_2) and also for a case without any special progress support (E_0). Evaluation of this value when the communication time is less than 15 microseconds is far from good precision due to benchmark inaccuracy, so for these cases the rough estimation is shown with the sign: "~".

Results and future work

The presented IMB-ASYNC benchmark suite is an attempt to implement a reliable tool inherited from a well-known and well-tested IMB code for both point-to-point and collective MPI benchmarks. The demand of such a benchmark comes from both application level developers and message-passing middleware developers since the asynchronous message passing progress, implemented both in hardware and software is a headmost feature of any modern MPI implementation. The calculation/communication overlapping efficiency estimation method is offered.

The particular results of testing sessions on Lomonosov-2 may lead to a few conclusions:

i) MPICH_ASYNC_PROGRESS=1 option of Intel MPI library (at least of 2017.1 its version) at many test points gives extremely huge penalty, especially at lower message sizes and at higher scale. Moreover, during the experiments it was noticed, that the MPI performance when the MPICH_ASYNC_PROGRESS is enabled tend to be very variative from one execution to another. Worst cases are even 2 times worse than those given in this paper. This makes it very doubtful that there is any reasonable application of this mode in practice.

ii) Manual progression support of non-blocking point-to-point operations with periodical MPI_Test() calls doesn't change anything in asynchronous progression of this type of communication, or the changes are of no practical importance.

iii) There is no way to significantly improve the asynchronous progression of non-blocking collective operations over small messages (at least, MPI_Iallreduce()) with Intel MPI 2017.1 on Lomonosov-2 supercomputer. This fact makes serious impact on parallel algorithms choice for new code which is created to be run on Lomonosov-2 system.

It is worth estimating the same metrics for Intel MPI 2019.2 and OpenMPI versions 2, 3 and 4 which are also installed on Lomonosov-2 system to see the full picture. The comparative tests on some other HPC systems are also in our plans.

Acknowledgements

The presented work is supported by the RSF grant No. 18-71-10075. The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University.

References

1. OSU Micro-Benchmarks, http://mvapich.cse.ohio-state.edu/benchmarks

2. Intel MPI benchmark, https://software.intel.com/en-us/articles/intel-mpi-benchmarks

3. IMB-ASYNC benchmark suite, https://github.com/a-v-medvedev/mpi-benchmarks

4. B. Krasnopolsky, The reordered BiCGStab method for distributed memory computer systems, Procedia Computer Science 1 (1) (2010) 213–218, ICCS2010. doi:10.1016/j.procs.2010.04.024.

5. Brinskiy, M., Supalov, A., Chuvelev, M., Leksikov, E.: Mastering performance challenges with the new MPI-3 standard. The Parallel Universe 18(1), 33–40 (2014), https://software.intel.com/sites/default/files/managed/6a/78/parallel_mag_issue18.pdf

2 nodes (28 ranks):	**4**	**128**	**2048**	**32768**	**524288**
Pure communication time, blocking MPI (T_{sync})	6,8	7,3	42,5	622,5	10132,5
Calculation time	10,0	10,0	100,0	1000,0	20000,0
Communication overhead, no special progress measures (T_{over0})	3,0	2,0	8,9	90,6	1819,0
Efficiency of overlapping, manual progress (E_0)	**~50%**	**~70%**	**79%**	**85%**	**82%**
Communication overhead, manual progress (T_{over1})	3,0	2,1	9,0	91,3	153,1
Efficiency of overlapping, manual progress (E_1)	**~50%**	**~70%**	**79%**	**85%**	**98%**
Communication overhead, MPICH_ASYNC_PROGRESS (T_{over2})	42,8	38,3	73,9	540,8	8098,2
Efficiency of overlapping, MPICH_ASYNC_PROGRESS (E_2)	**-530%**	**-425%**	**-74%**	**13%**	**20%**
16 nodes (224 ranks):					
Pure communication time, blocking MPI (T_{sync})	7,9	13,4	186,0	2927,1	46630,4
Calculation time	10,0	10,0	200,0	5000,0	100000,0
Communication overhead, no special progress measures (T_{over0})	3,5	5,0	17,7	153,2	2436,8
Efficiency of overlapping, manual progress (E_0)	**~50%**	**~60%**	**90%**	**95%**	**95%**
Communication overhead, manual progress (T_{over1})	3,3	5,1	16,6	153,2	1575,6
Efficiency of overlapping, manual progress (E_1)	**~50%**	**~60%**	**91%**	**95%**	**97%**
Communication overhead, MPICH_ASYNC_PROGRESS (T_{over2})	55,4	47,4	87,0	2300,4	45772,0
Efficiency of overlapping, MPICH_ASYNC_PROGRESS (E_2)	**-603%**	**-255%**	**53%**	**21%**	**2%**
64 nodes (896 ranks):					
Pure communication time, blocking MPI (T_{sync})	8,0	13,4	184,8	2919,1	47465,5
Calculation time	10,0	20,0	500,0	5000,0	100000,0
Communication overhead, no special progress measures (T_{over0})	6,3	8,0	143,4	1442,8	27571,3
Efficiency of overlapping, manual progress (E_0)	**~20%**	**~40%**	**22%**	**51%**	**42%**
Communication overhead, manual progress (T_{over1})	6,4	8,1	147,3	1483,7	27428,5
Efficiency of overlapping, manual progress (E_1)	**~20%**	**~40%**	**20%**	**49%**	**42%**
Communication overhead, MPICH_ASYNC_PROGRESS (T_{over2})	61,3	61,4	727,0	5147,2	95445,1
Efficiency of overlapping, MPICH_ASYNC_PROGRESS (E_2)	**-665%**	**-356%**	**-293%**	**-76%**	**-101%**

Table 1. IMB-ASYNC sync_pt2pt, async_pt2pt results. Point-to-point pairwise communications, average time among all pairs in MPI_COMM_WORLD. The first row shows the amount of elements in each MPI message. Message element is MPI_DOUBLE. All times are given in microseconds.

2 nodes (28 ranks):	**4**	**128**	**2048**	**32768**	**524288**
Pure communication time, blocking MPI (T_{sync})	12,0	15,0	46,0	308,9	7221,7
Calculation time	20,0	20,0	100,0	500,0	20000,0
Communication overhead, no special progress measures	12,5	15,3	46,0	388,2	7241,4
Efficiency of overlapping, manual progress (E_0)	**~0%**	**-2%**	**0%**	**-26%**	**0%**
Communication overhead, manual progress (T_{over1})	12,7	15,7	41,0	365,7	6913,5
Efficiency of overlapping, manual progress (E_1)	**~0%**	**-4%**	**11%**	**-18%**	**4%**
Communication overhead, MPICH_ASYNC_PROGRESS (T_{over2})	231,4	278,9	290,8	216,9	1534,1
Efficiency of overlapping, MPICH_ASYNC_PROGRESS (E_2)	**-1828%**	**-1755%**	**-532%**	**30%**	**79%**
16 nodes (224 ranks):					
Pure communication time, blocking MPI (T_{sync})	36,6	35,9	81,1	547,9	8184,8
Calculation time	50,0	50,0	200,0	1000,0	20000,0
Communication overhead, no special progress measures	37,6	39,9	84,5	551,2	7718,8
Efficiency of overlapping, manual progress (E_0)	**-3%**	**-11%**	**-4%**	**-1%**	**6%**
Communication overhead, manual progress (T_{over1})	37,0	39,6	69,1	458,3	7331,3
Efficiency of overlapping, manual progress (E_1)	**-1%**	**-10%**	**15%**	**16%**	**10%**
Communication overhead, MPICH_ASYNC_PROGRESS (T_{over2})	863,3	875,7	834,8	192,8	3136,7
Efficiency of overlapping, MPICH_ASYNC_PROGRESS (E_2)	**-2257%**	**-2339%**	**-929%**	**65%**	**62%**
64 nodes (896 ranks):					
Pure communication time, blocking MPI (T_{sync})	60,6	58,7	107,5	655,8	9192,8
Calculation time	100,0	100,0	200,0	1000,0	20000,0
Communication overhead, no special progress measures (T_{over0})	60,4	57,3	139,5	823,6	7585,6
Efficiency of overlapping, manual progress (E_0)	**0%**	**2%**	**-30%**	**-26%**	**17%**
Communication overhead, manual progress (T_{over1})	55,8	55,9	136,4	725,3	7508,4
Efficiency of overlapping, manual progress (E_1)	**8%**	**5%**	**-27%**	**-11%**	**18%**
Communication overhead, MPICH_ASYNC_PROGRESS (T_{over2})	617,8	626,6	778,1	1262,1	21514,5
Efficiency of overlapping, MPICH_ASYNC_PROGRESS (E_2)	**-920%**	**-968%**	**-624%**	**-92%**	**-134%**

Table 2. IMB-ASYNC sync_allreduce, async_allreduce results. MPI_Allreduce, MPI_Iallreduce collective operation on the full MPI_COMM_WORLD. The first row shows the amount of elements in each MPI message. Message element is MPI_DOUBLE. All times are given in microseconds.

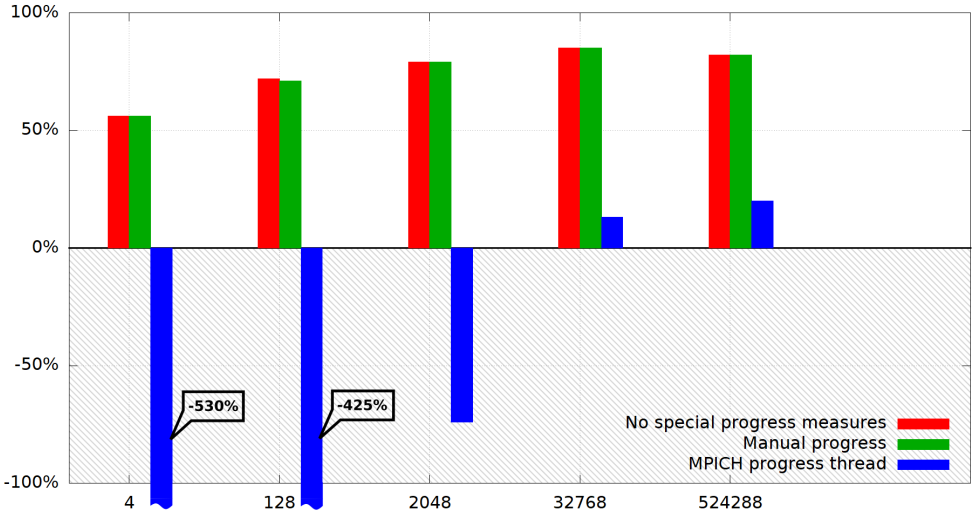

Figure 1. Overlapping efficiency: 2 nodes, pt2pt benchmark in various modes.

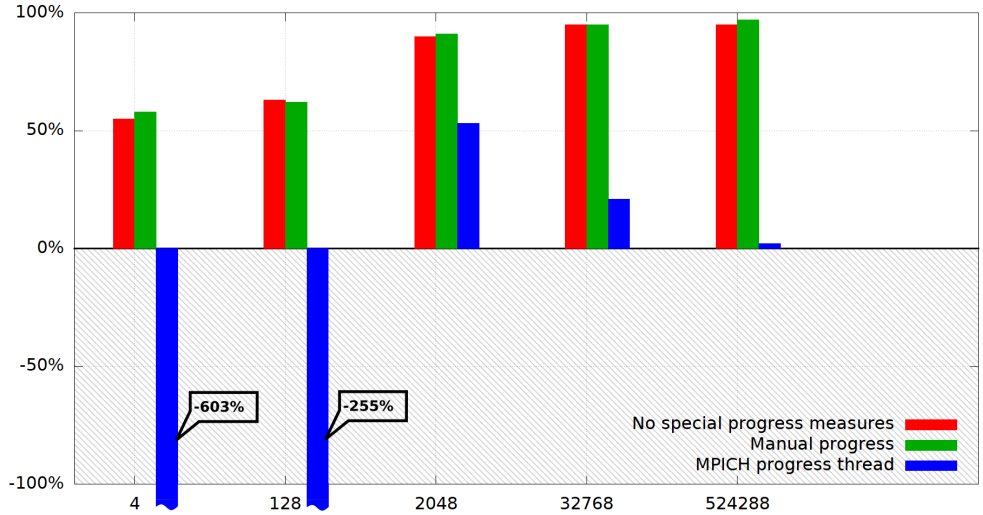

Figure 2. Overlapping efficiency: 16 nodes, pt2pt benchmark in various modes.

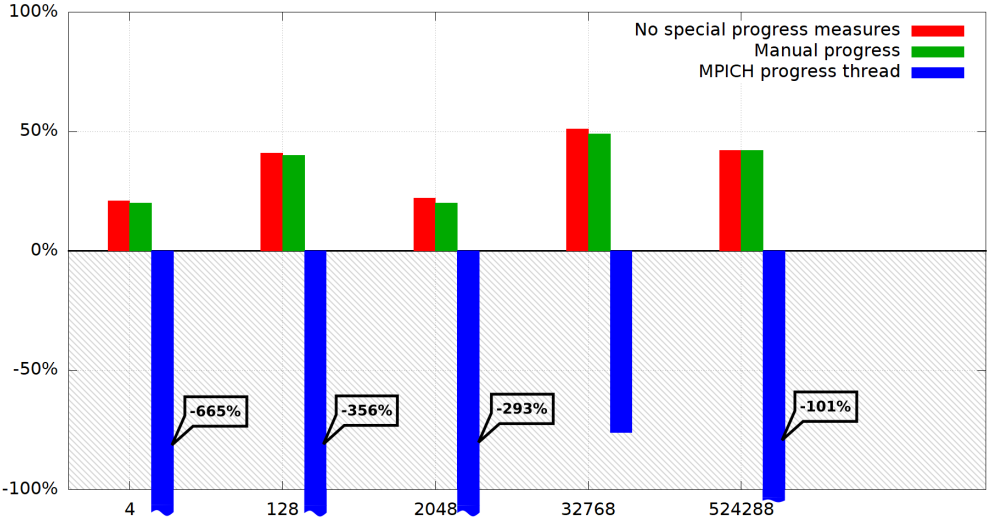

Figure 3. Overlapping efficiency: 64 nodes, pt2pt benchmark in various modes.

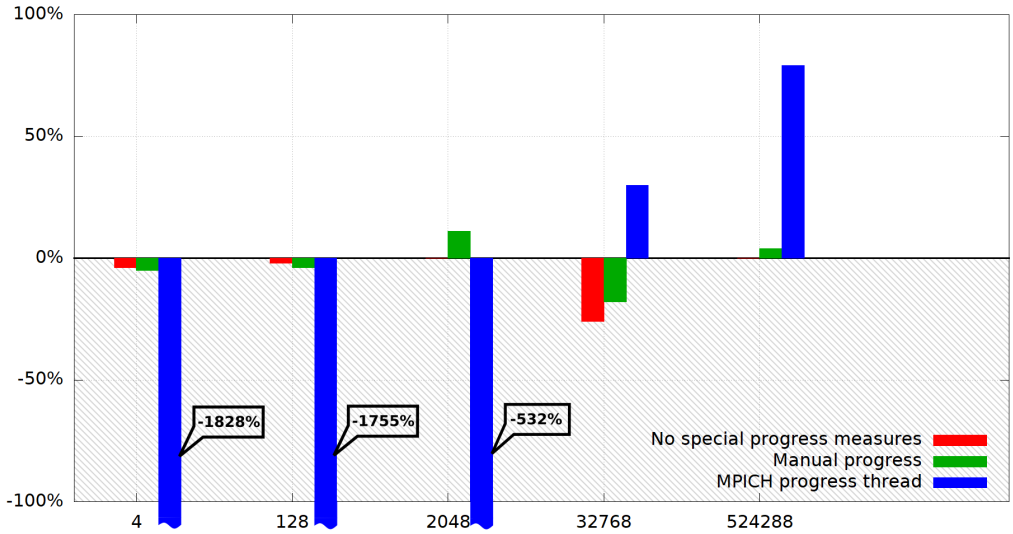

Figure 4. Overlapping efficiency: 2 nodes, pt2pt benchmark in various modes.

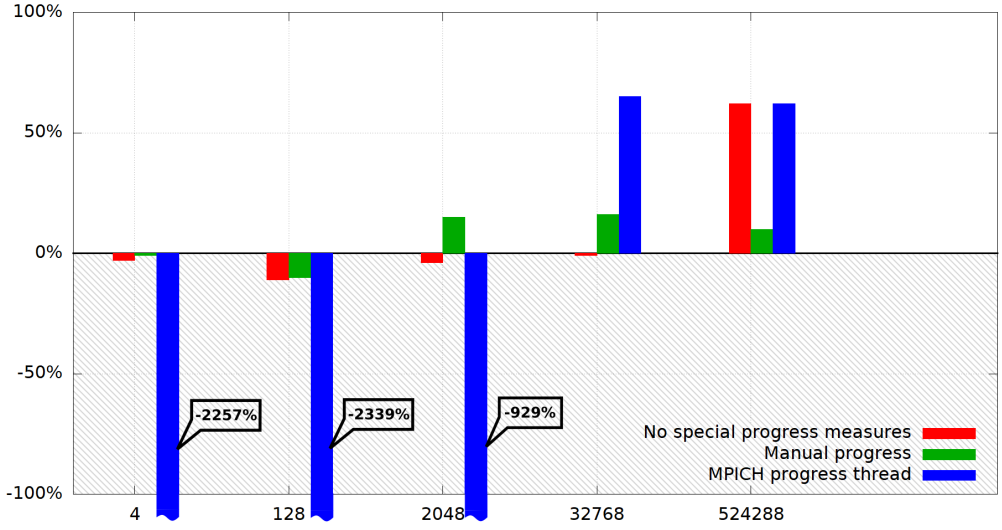

Figure 5. Overlapping efficiency: 16 nodes, pt2pt benchmark in various modes.

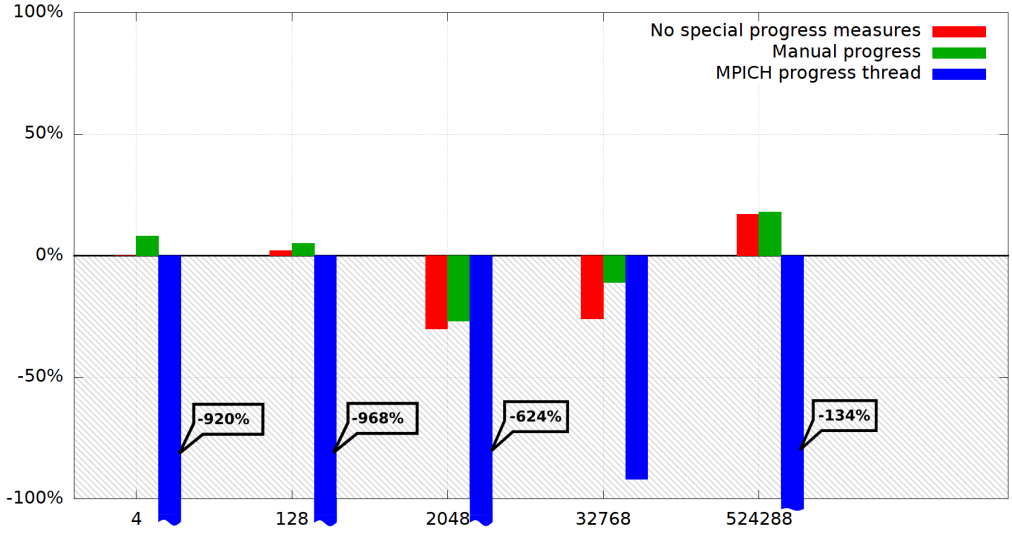

Figure 6. Overlapping efficiency: 64 nodes, pt2pt benchmark in various modes.