Optimizing Remote Data Transfers in X10 Evaluation of AT-Opt for single node set-up

ACM Reference Format:
. 2018. Optimizing Remote Data Transfers in X10 Evaluation of AT-Opt for single node set-up. In *Proceedings of ACM Conference (PACT'18).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 IMPLEMENTATION AND EVALUATION

In this section, we evaluate our proposed optimization AT-Opt on two different systems - a one node Intel system, where each node has two Intel E5-2670 2.6GHz processors, with 16 cores per processor and 64GB RAM; and a one node AMD system, where each node has a AMD Abu Dhabi 6376 processor containing 16 cores per processor, with 512GB RAM.

We implemented AT-Opt in the x10v2.6.0 compiler x10c (Java backend) and x10c++ (C++ backend). Based on the ideas from the insightful paper of George et al., we report the execution times by taking a geomean over thirty runs.

We evaluated AT-Opt using 12 benchmark kernels from IMSuite: breadth first search (BF - computes the distance of every node from the root and DST - computes the BFS tree), byzantine consensus (BY), routing table creation (DR), dominating set (DS), maximal independent set (MIS), committee creation (KC), leader election (DP - for general network, HS - for bidirectional ring network, and LCR - for unidirectional ring network), spanning tree (MST) and vertex coloring (VC). We also studied many other benchmarks made available in the X10 distribution, but none of them met our selection requirements: (a) presence of at-construct in the program, and (b) de-reference of object (other than distributed arrays) fields at the remote place.

Note on choice of input size: For all the benchmarks kernels, the chosen input size was the largest input such that on our 32-core Intel system (64 GB RAM), when the corresponding program is run by setting X10_NPLACES=2, it does not take more than an hour to execute and does not run out-of memory. Such larger input help expose the overheads in our approach better. We executed the chosen kernels on the specified inputs by varying the number of places (in powers of two) and threads per place such that at any point of time the total number of threads (= #places × #num-threads-per-place) is equal to the number of cores. This is achieved by setting the runtime environment variable X10_NPLACES and X10_NTHREADS (threads per place) appropriately.

PACT'18, November 2018, Limassol, Cyprus

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1.1 Evaluation of AT-Opt in x10c (Java) Backend

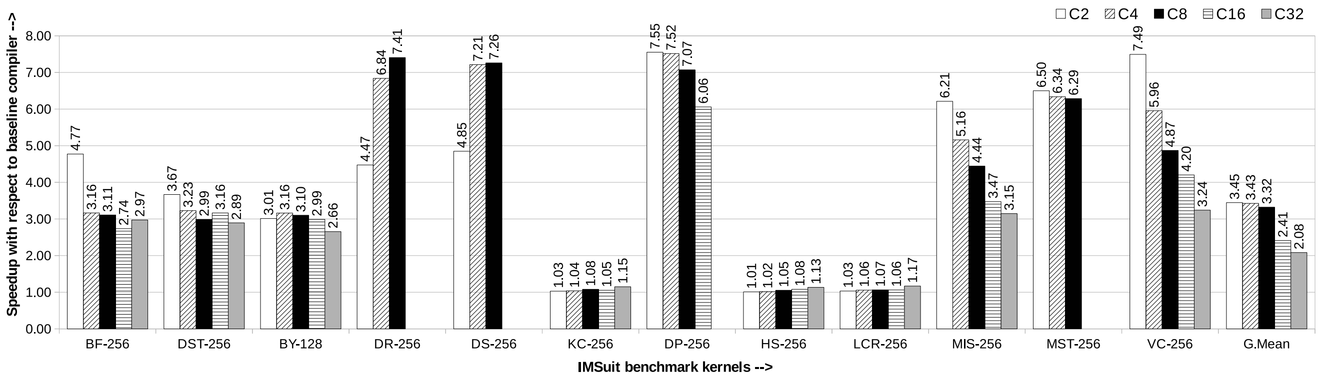
We report experimental results for two cases: (a) Base - the baseline version without any communication optimizations; (b) AT-Opt - the optimized version that uses the techniques described in this paper.

Figure 1a shows the speedups achieved by the kernels on the Intel system for varying number of places and threads. In the context of Base: DR, DS and MST ran out of memory with 16 and 32 places and DP ran out of memory with 32 places. We can see that the AT-Opt leads to significantly large speedups (geometric mean of 2.88×). As the number of places increase, while the amount of communicated data remains the same (and hence gains because of AT-Opt), there is an increase in the amount of inter-place communication and thus the speedups reduce slightly (but still remain significant).

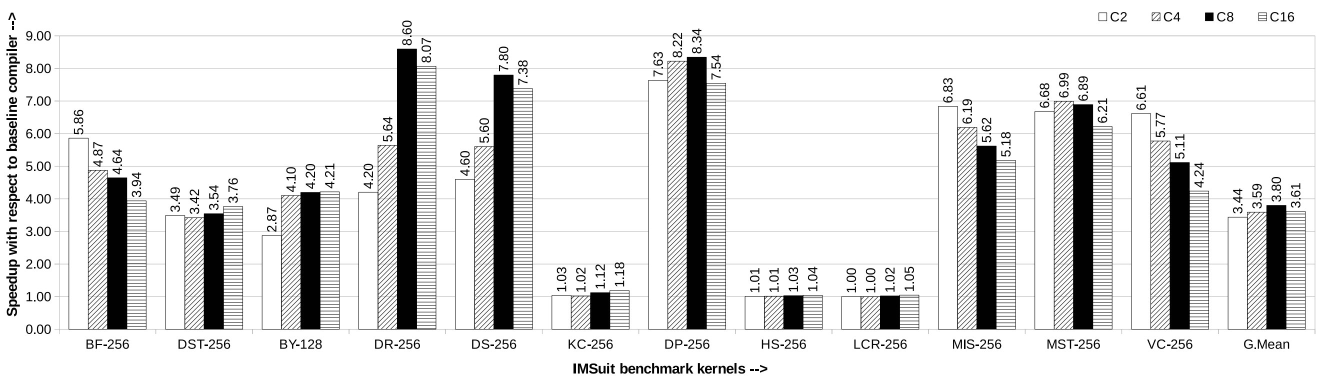
Note that for programs like DR, DS, and BY, the obtained speedups for C_4 and C_8 are significantly better than C_2. This is because, for both Base and AT-Opt the programs run much slower (more inter-place communication) in configuration C_2 compared to their counterparts in the configurations C_4 and C_8. This is due to the way in which the input graph is distributed among the nodes.

For kernels KC, HS and LCR, the speedups are not substantial. This is due to the amount of data getting communicated across places (in Base, itself) is very less; consequently the reduction in the communicated data is also less (order of few hundred MBs). For the rest of the benchmarks, AT-Opt leads to significant amount of gains in the execution time (in line with the reduction in the communicated data).

Figure 1b shows the speedups achieved by the kernels on the AMD system for varying number of places and threads. It can be seen that with respect to Base, the AT-Opt optimizer achieved large speedups (geometric mean of 3.61×). As it can be seen, the gains resulting because of AT-Opt are similar to those in the Intel system.



(a) Speedups on an Intel system; totalCores=32.



(b) Speedups on an AMD system; totalCores=16.

Figure 1: Speedups for varying number of places (#P) and threads (#T). Configuration C_i denotes #P=i and #T=totalCores/i; Speedup = (execution time using `Base` / execution time using `AT-Opt`). In x10c (Java) backend.

Note that in case of the AMD systems, none of the benchmarks (including DR, DS, MST, and DP) ran out of memory during execution for any number of places (when compiled using the `Base`); this is because of the large available memory in the AMD system (512 GB, compared to 64 GB on the Intel system).

It is encouraging to note that `AT-Opt` optimized codes could run on both the systems (without going out of memory), even with much larger inputs (in the order of thousands of nodes). This shows that the impact of `AT-Opt` is not only in reducing the execution time, but also in making the programs scalable by reducing the memory requirements significantly.

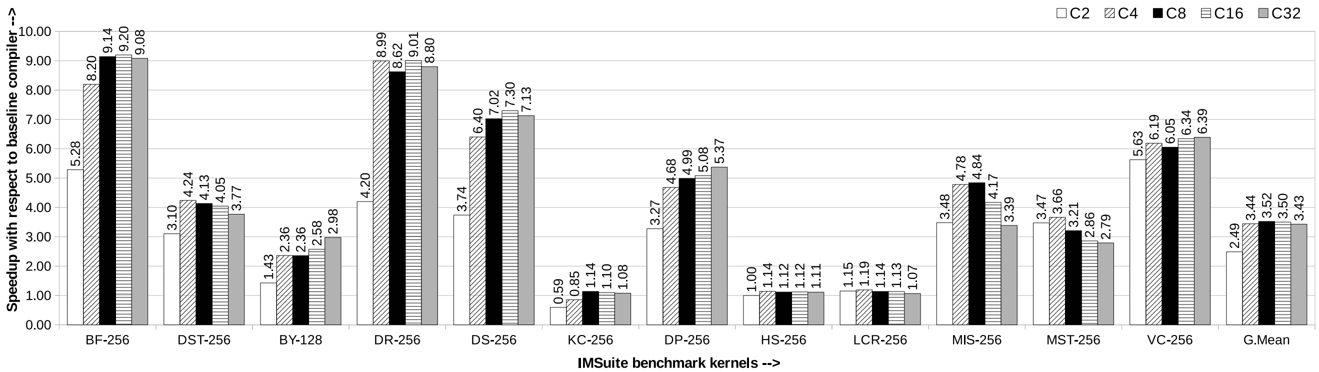
1.2 Evaluation of `AT-Opt` in x10c++ (Cpp) Backend

Figure 2a shows the speedups achieved by the kernels on the Intel system for varying number of places and threads in x10c++ backend. We can see that the `AT-Opt` leads to significantly large speedups (geometric mean of 3.25×). For kernels other than MIS and MST, as the number of places increases from C_2 to C_{32} the speedups increases considerably.

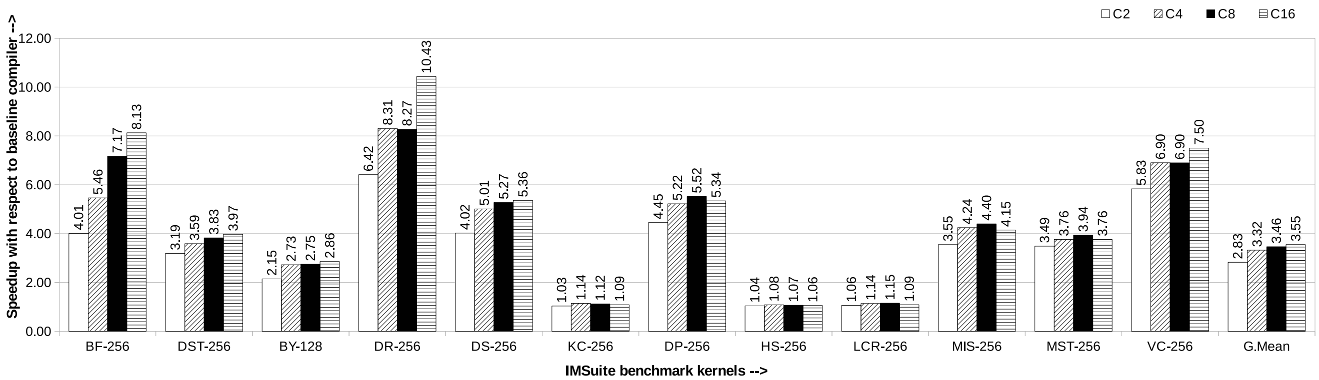
Similar to x10c (Java) backend, for kernels KC, HS and LCR, the speedups are not substantial. This is due to the amount of data getting communicated across places (in `Base`, itself) is very less; consequently the reduction in the communicated data is also less. For the rest of the benchmarks, `AT-Opt` leads to significant amount of gains in the execution time (in line with the reduction in the communicated data).

Figure 2b shows the speedups achieved by the kernels on the AMD system for varying number of places and threads. It can be seen that with respect to `Base`, the `AT-Opt` optimizer achieved large speedups (geometric mean of 3.28×). Here, for all kernels, as the number of places increases from C_2 to C_{16} the speedups increases considerably. As it can be seen, the gains resulting because of `AT-Opt` are similar to those in the Intel system (Figure 2a).

Summary: We have studied the benchmarks and their behavior carefully and found that the actual amount of speedup varies depending on multiple factors: (1) Number of `at-constructs`



(a) Speedups on an Intel system; totalCores=32.



(b) Speedups on an AMD system; totalCores=16.

Figure 2: Speedups for varying number of places (#P) and threads (#T). Configuration C_i denotes #P=i and #T=totalCores/i; Speedup = (execution time using `Base` / execution time using `AT-Opt`). In x10c++ (Cpp) backend.

executed. (2) Amount of data getting serialized during each communication. (3) Amount of other components of remote communication (meta-data such as runtime-type information, data related to the body of the `at-construct`, and so on) (4) Time taken to perform inter-place communication. (5) The nature of the input, runtime/OS related factors and the hardware characteristics. While the factor (2) is the only one that is different between `Base` and `AT-Opt` optimized codes, the impact of factor (2) can be felt on (4) as well. Since `AT-Opt` helps reduce the factors (2) (and consequently factor (4)) it leads to significant performance gains.