Sentinel-2 Level-2 Processing with Sen2Cor
An Approach for Performance Improvement

Taylan Özden[1], Uwe Müller-Wilm[1], Jérôme Louis [2], Bringfried Pflug[3],
Valentina Boccia[4], Ferran Gascon[4], Rosario Quirino Iannone[5], Roberto de Bonis[5]

(1) Telespazio VEGA - A Leonardo / Thales Company (Germany),
Email: taylan.oezden@telespazio-vega.de,
(2) Telespazio France, (3) German Aerospace Centre, Remote Sensing Technology Institute,
(4) European Space Agency, ESRIN, (5) Rhea spa, Italy.

In June 2017, ESA started systematic Level-2A processing of Sentinel-2 acquisitions over Europe using the Cloud Screening and Atmospheric Correction Processor named Sen2Cor. Since March 2018, Level-2A products are generated by the official Sentinel-2 ground segment (PDGS) and are available on the Copernicus Open Access Hub. Although, the embedding of Sen2Cor in the processing chain has been proven quite satisfactory and several optimizations have already been performed up to the recent Version 2.8, there is still a desire to improve its runtime performance, in order to warrant the availability of Level-2A data with a faster update ability. The objective of the study presented is to develop and test several strategies to improve the processing of Sen2Cor using compilation, parallel, distributed and cloud-based processing techniques.

LEVERAGING THE INTEL® MATH KERNEL LIBRARY FOR SCIENTIFIC PYTHON

Scientific Python libraries such as SciPy and NumPy which are being used extensively in the Sen2Cor processor benefit mainly from mathematical and specifically linear algebra routines tied close to the underlying hardware architecture. In these instances, the Basic Linear Algebra Subprograms (BLAS) as well as the Linear Algebra Package (LAPACK) are being utilized.

Libraries relying on heavy numerical linear algebra routines mostly conform these specifications and are therefore BLAS- and LAPACK-compatible. In our work, we propose to ship the Intel® Math Kernel Library (MKL) as the default core library being used by the Sen2Cor processor.

Intel MKL:

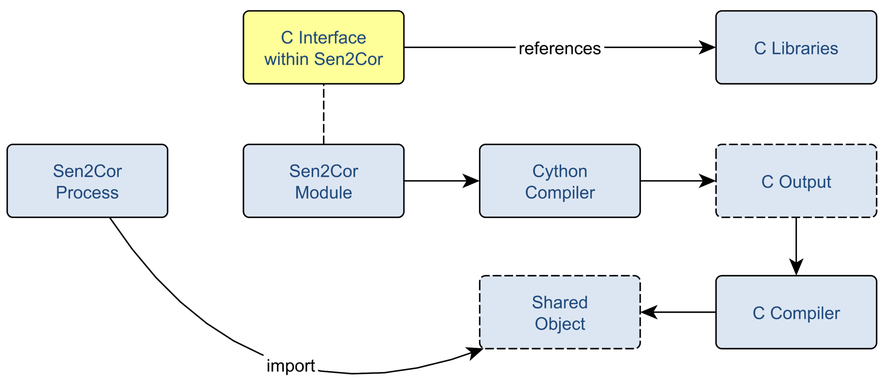
The Intel MKL provides core mathematical routines conforming BLAS and LAPACK which are in addition highly optimized for Intel processors. Currently, the PDGS is based on an Intel architecture and the Sen2Cor processor is expected to benefit from the Intel MKL on the PDGS.

Results of performance tests conducted on similar architectures based on Intel commodity hardware has proven the Intel MKL's advantages over alternative approaches. Our performance tests included desktop systems and virtual machines as well as different server architectures.

COMPILATION AND DISTRIBUTED MEMORY PARALLELIZATION

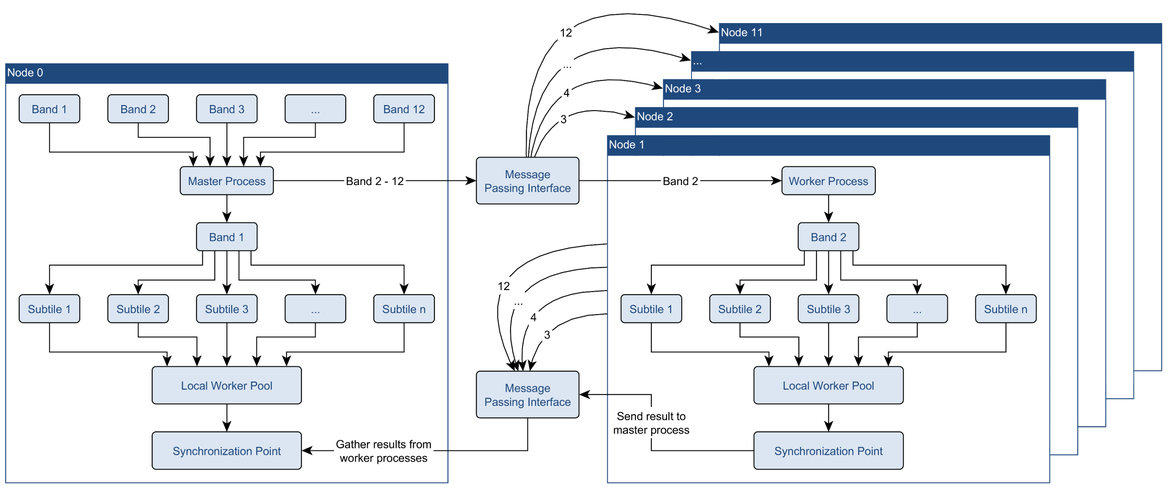
Python applications do often benefit from compilation. Although it is hard to improve performance by compiling Python applications which already make heavy use of precompiled libraries, we investigated in compilation approaches for the Sen2Cor processor to statically link low-level C routines.

Our approach follows a similar procedure which already is being used within the Sen2Cor. We utilize the Cython compiler to generate precompiled shared objects and extend the current approach through a C interface within the Sen2Cor processor.



The parallelization approach for distributed memory architectures relies on the independent processing of different bands. Similar to our shared memory approach, bands are being distributed among different nodes. Every node is parallelizing the processing of sub-tiles on its shared memory architecture, thus resulting in hybrid parallelization.

This approach is suitable for high-performance and cloud computers capable of passing messages between different nodes.



SHARED MEMORY PARALLELIZATION

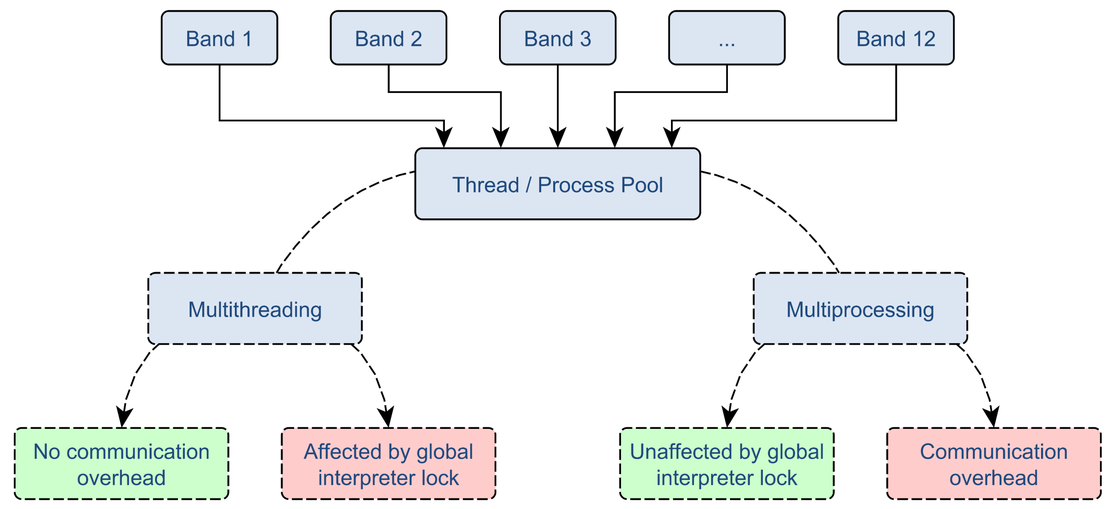
The approach for shared memory parallelization takes compute-intensive sections into account and parallelizes them by leveraging either multiple threads or processes. Key optimizations have already been successfully applied in the retrieval of surface reflectance. Each worker task is processing one band independently while synchronizing on critical sections.

Multithreading approach:

The multithreading approach is using threads within one single process to parallelize the processing of different bands. This approach suffers from the global interpreter (GIL) lock of Python which prevents multiple threads from executing Python bytecode at one time. However, since the surface reflectance retrieval is utilizing low-level routines which are able to release the GIL, an optimization of the performance can be observed.

Multiprocessing approach:

The multiprocessing approach will utilize multiple processes to process bands independently. While processes are not subject to the GIL, they are not as lightweight as threads. Furthermore, the communication overhead needs to be taken into account, since processes do not share the same virtual address space. Additionally, unnecessary data copies need to be avoided.

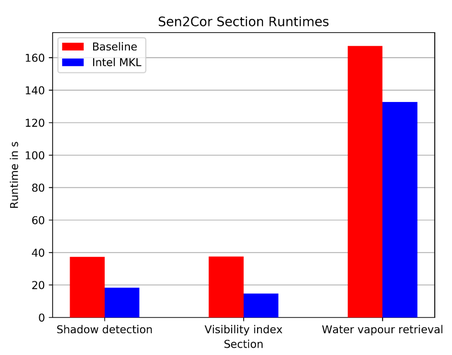


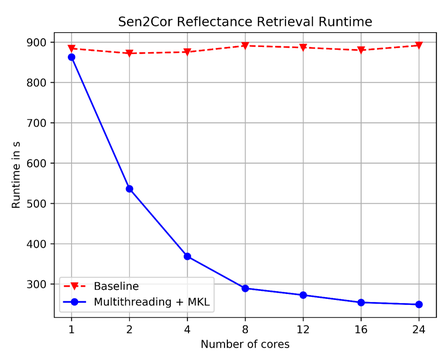
CONCLUSION AND RECOMMENDATIONS

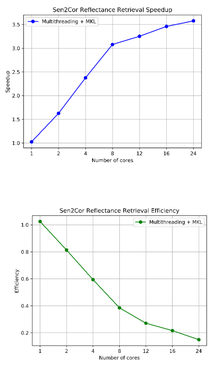
The conducted performance tests have demonstrated the potential and capabilities of our multithreading approach powered by the Intel MKL in compute-intensive sections of the Sen2Cor processor.

We recommend to process time-critical parts of the algorithm in parallel by using a multithreaded backend and shipping the Intel MKL as the core library for routines conforming BLAS and LAPACK. Further tests need to be conducted to observe performance benefits by using a multiprocessed backend for the shared memory parallelization. We expect the multiprocessed backend to have a better scalability than the multithreaded approach.

Additionally, preliminary results have shown that interfacing directly with C libraries can benefit the performance of Sen2cor. The parallelization approach for shared memory architectures also has proven that independent processing of compute-intensive functions is possible and is expected to benefit the performance on distributed memory architectures.







The figures above demonstrate the benefits of using the Intel MKL and our multithreaded approach within different sections of the Sen2Cor processor. Although our multithreaded backend is limited by Python's global interpreter lock, it has shown its capabilities of performance improvements on multi-core architectures. Additionally, we were able to show that oversubscription of available cores also influences the performance positively.