Catalysts
software is our passion

GRASP/3MI Assessment Report

Final Version

Michael Aspetsberger, Stefan Amberger, Daniel Marth — Catalysts GmbH
Pavel Litvinov, David Fuertes, Oleg Dubovik — GRASP SAS

GRASP-3MI-Study_D1_Assessment-Report_final — 19.10.2018

Table of Contents

1 Introduction

1.1 Purpose and Scope

This document summarizes the assessment of GRASP for its application to the EPS-SG/3MI instruments with regard to computational performance, interfacing with the future ground segment, and its suitability for formal specification and documentation. The scientific quality and suitability of the result parameters is not part of this study.

This document is composed of this introductorily chapter, including background and general notes, followed by one chapter for each of the three assessments, and finally drawing the overall conclusions for this study. The appendices include detailed assessment output for additional information.

Throughout this document, common abbreviations are used which are assumed to be known to the reader.

1.2 Background of this Study

The Multi-Viewing Multi-Channel Multi Polarisation Imager (3MI) on board EPS-SG/MetOp-SG A will provide excellent capacities for observing aerosol and surface aspects of our planet. Having up to 14 viewing angles and polarimetric observations available distinguishes it from all other currently flying instruments. This potential is also a burden. Traditional aerosol retrieval algorithms that rely on lookup tables are not able to fully exploit all observations: the more measurements are available, the larger the dimension and the bigger the look-up tables. Resorting to a reduced result quality is a weak compromise which wastes resources.

For its EPS-SG and MTG missions, EUMETSAT has selected a novel approach to handling the payload data ground segment processing: Instead of the traditional batch-processing workflow, where a chunk of the payload data is being transformed in a step-by-step manner, a stream-processing has been adopted, where the chunk is broken down and the processing steps occur interleaved.

The GRASP algorithm has demonstrated on several occasions that it is able to significantly outperform traditional and competing algorithms. Specifically, for PARASOL/POLDER, which can be considered an ancestor of 3MI, exceptional result quality has been achieved. However, there are still concerns in the community of the maturity of GRASP in terms of computational intensity and documentability. Likewise, there are concerns on how GRASP can interface with the future EPS-SG PDAP.

The objective of this study was to assess three questions:

1. Is GRASP fast enough to run as operational 3MI product or can it be improved enough?
2. How can GRASP be implemented in the EPS-SG PDAP or what would need to change?
3. Can GRASP be specified/documented to be eventually implemented?

The study has been carried out on behalf of EUMETSAT by Catalysts GmbH and GRASP SAS.

1.3 GRASP Configurations

In the frame of this study, several configurations of GRASP have been tested. These configurations have been generated by GRASP SAS and CNRS/LOA in the frame of different EUMETSAT studies and were provided as input to this study.

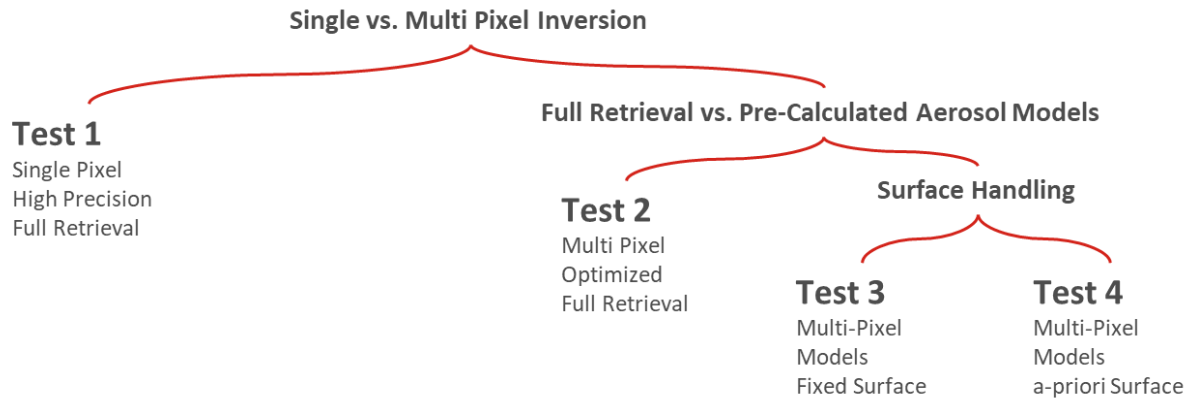


Figure 1: Overview of the GRASP Settings

The 4 provided configurations are depicted in Figure 1 and can be summarized as:

1. **test1.sd.sp.hp.8wl.full_retrieval**
   a single-day, single-pixel, high precision configuration that retrieves surface- and atmospheric parameters

2. **test2.sd.mp.opt.8wl.full_retrieval**
   a single-day, multi-pixel, optimized configuration that retrieves surface- and atmospheric parameters

3. **test3.sd.mp.models.8wl.fix_surf**
   a single-day, multi-pixel configuration using pre-calculated models that retrieves atmospheric parameters, assuming a fixed surface climatology

4. **test4.sd.mp.models.8wl.apri_surf**
   a single-day, multi-pixel configuration using pre-calculated models that retrieves surface- and atmospheric parameters, using an initial-guess surface climatology

All configurations are single-day observations. While 3 out of the 4 feature multi-pixel retrievals, these only apply to the spatial domain, not the temporal. As such they are perfectly suitable for near-real-time processing.

1.4 Test Data Sets

The test data used in this study has been provided by EUMETSAT, and represents 1 simulated orbit:

- W_xx-eumetsat-darmstadt,SAT,SGA1-3MI-1C-REF_C_EUMT_xxxxxxxxxxxxx_G_D_20080223085157_20080223085241_T_N____.nc

The orbit comes in 3 variations with varying parallax correction:

- L1C_XML18_TDSv4_noparallax
- L1C_XML18_TDSv4_parallax
- L1C_XML18_TDSv4_parallaxCloud

2 Performance Assessment

2.1 Introduction

Whenever an algorithm is to be integrated as operational processor, it needs to operate reliably in terms of quality and speed. In the frame of the 3MI mission GRASP is being evaluated as operational algorithm and this report summarizes the performance assessment of GRASP and several retrieval scenario settings for a potential NRT processing setup. The key questions to be answered are:

- Is GRASP fast enough to run as operational 3MI NRT processor?
- What processing resources are required to run GRASP in NRT?
- How can GRASP be optimized further?

2.2 Approach

In order to determine if GRASP is suitable given these requirements, we conducted benchmarks, testing different GRASP configurations on a simulated orbit of 3MI data that has been provided by EUMETSAT. All configurations work with data from a single measurement time only ("NRT configurations"), as opposed to configurations that would make use of multiple satellite overpasses over time.

The benchmarks were conducted in a controlled environment, with multiple runs to be able to collect the necessary statistics on performance for extrapolation and to estimate the necessary processing cores.

Finally, we profiled GRASP on a representative subset of this orbit, to find hot-spots in the code that can possibly be optimized. For each we estimated the benefit from speed-up and how this relates to the overall processing time.

2.3 Constraints

As per the information provided at the Kick-off[1], the following constraints apply to the processing time:

- 3MI L2 PGF shall receive one chunk equivalent to 3 minutes sensing time each 85 seconds.
- 3MI L2 PGF shall be able to process each received chunk in less than 469 seconds.

Consequently, the end-to-end constraint for the Level 2 Aerosol Product for a 100 minute orbit (50 min of sensing time) will be 7817 seconds or 130 minutes. Accounting for the various I/O overheads, the product formatting, and the co-registration, effectively a total of about 45 minutes processing time are considered to remain for the aerosol processor. It is understood that the processing happens in chunks, not full orbits.

The physical hardware foresees no Graphics Processing Units but only standard x86_64 cores. There are no clearly defined constraints as to how many cores are available for a given product. Following the discussions in

[1] GRASP-3MI-Study_Meeting_2018-04-11_KO-PDAP-constraints.pptx

the Sentinel-4 Level 2 operational setup, a general guideline asks for no more than 100 cores to be used for the NRT-processing.

2.4 Setup

This section describes the key qualities of the testing setup. This includes the used hardware, the test dataset used, the scheduling solution that powered the benchmarks, and repeatability concerns.

2.4.1 Hardware

The benchmarks were run on a single node with the following hardware

Processor

Intel(R) Xeon(R) CPU E5-2650 v4
2 sockets with 12 physical cores each
30 MB L3 cache
156 KB L2 cache
max 2.9 GHz

Memory

8x 32 GB DDR4 DIMM (in total 256 GB)
2400 MHz configured, 2667 MHz top speed

Storage

500GB Micron_5100_MTFD SSD

2.4.2 Benchmark Test Dataset

No significant impact has been detected for using either of the parallax corrections with regard to the performance. Unless otherwise noted, the parallaxCloud variant has been used, given that this is the most representative one.

2.4.3 Scheduling

To mimic a EUMETSAT production environment as good as possible, we took certain precautions in data locality and scheduling.

The simulated 3MI orbit was located on the local SSD. To avoid hyper-threading, all benchmarks were executed on no more than 24 cores simultaneously. At the same time we aimed to create realistic benchmarks by utilizing the node as much as possible while staying out of hyper-threading. Benchmarks that only utilize part of the CPU and not all physical cores result in faster runs, since the CPU can keep a single utilized core cool at a faster clock-speed than it could if all cores were generating heat.

2.4.4 Instrumentation

To derive performance statistics, we used code instrumentation to report timing information, which we parsed from log files, and aggregated for further generation of statistics.

GRASP already measures and logs general data-set parameters, as well as timing information for retrievals of segments (groups of pixels) of data. We parsed the relevant sections of the logs to create the necessary statistics.

This allows us to benchmark only those sections of the code, that are relevant for EUMETSAT: those that run the algorithms, not those that do IO or data preparation, which will be handled by EUMETSATs PDAP processing environment.

2.4.5 Repeatability

Benchmarks need to be repeatable to be useful. We took care of repeatability by placing all manual commands, inputs and configurations of the benchmarks as well as of GRASP in configuration files. These were the input to a program that reliably executes the benchmarks in exactly the same way. A total of 18 iterations have been completed, showing a variability of less than 1 %.

GRASP Setup

Due to the use of static arrays of configurable length, multiple GRASP versions were installed on the node. Installation happened via scripts, such that even these installations can be repeated, if required. The versions of GRASP and its sub-modules were logged for each benchmark using the `git describe --tags --always --dirty` command. This command returns the last tag, the commits since the last tag, and the commit hash of the current commit. This gives a precise version, as well as a good overview for humans to read. It also ensures that the code that is used in a benchmark is checked in, and not local code that has been worked on.

The version of GRASP and its configuration used throughout the tests is shown in Table 1.

Table 1: GRASP versions used in this project.

Component	Version
GRASP	v0.8.1-4 (git hash 6a04035)
constants-set-mmmi	v0.8.1.0 (git hash ced4f17)
mmmi-official-tests (settings)	V1.2.1 (git hash 9c87ab5)

Benchmark configurations

Since benchmarks themselves can have high complexity, they were specified in configurations that were passed to a scheduler to extrapolate and execute.

The complexity of benchmarks varies between calling a single binary on a single input-file and calling multiple binaries on hundreds of input-files, with different command line arguments, configurations and configuration overrides. This complexity is handled by benchmark configurations that allow to

- persist the specifics of a benchmark
- call a benchmark again on the same specifics

An example for a benchmark configuration is:

```
retrievals:
- name: 2x2_test3.sd.mp.models.8wl.fix_surf
  settings: ../mmmi-official-tests/test3.sd.mp.models.8wl.fix_surf/run.yml
  binary: ../src/grasp-installations/v0.8.1-2x2-plots/bin/grasp_app
  direct_overrides:
    input.segment.t: "1"
    input.segment.x: "2"
    input.segment.y: "2"
    output.tile.function: csv
  expandable_overrides:
    input.file:
../3mi/ftp.eumetsat.int/pub/EPS/out/lang/3MI/L1C/L1C_XML18_TDSv4_parallaxCloud/*.nc
    input.transformer_settings.mmmi_surface_ardeco.climatology:
../3mi/oc.catalysts.cc/climatology/*.h5
- name: 20x20_test3.sd.mp.models.8wl.fix_surf
  settings: ../mmmi-official-tests/test3.sd.mp.models.8wl.fix_surf/run.yml
  binary: ../src/grasp-installations/v0.8.1-20x20-plots/bin/grasp_app
  direct_overrides:
    input.segment.t: "1"
    input.segment.x: "20"
    input.segment.y: "20"
    output.tile.function: csv
  expandable_overrides:
    input.file:
../3mi/ftp.eumetsat.int/pub/EPS/out/lang/3MI/L1C/L1C_XML18_TDSv4_parallaxCloud/*.nc
    input.transformer_settings.mmmi_surface_ardeco.climatology:
../3mi/oc.catalysts.cc/climatology/*.h5
```

Listing 1: Example of a benchmark configuration

In this example a two different configurations are called: one with segment size 2x2 and one with segment size 20x20, both using different GRASP binaries, but the same set of input files and the same GRASP settings.

GRASP settings

GRASP itself is highly configurable, with dozens of switches and parameters, that specify different input parameters, optimizations, constants sets, and mathematical models. Since these don't differ most of the time between runs on the same instrument, they are saved in "GRASP settings files". The configurations used are described in Chapter 1.3.

The reference to these settings files can be seen in the example above, where they are located in the field `retrievals[*].settings` in the benchmark configuration.

2.4.6 Profiling

Profiling was done by calling GRASP via the tool Valgrind[2] which dynamically instruments the binary without the necessity to re-compile. Analysis was carried out by opening the resulting "callgrind" files with Kcachegrind[3], a graphical tool that can interpret the information in said files.

Figure 2 and Figure 3 give an impression of Kcachegrind, and illustrates how the values in section "Results" were determined.

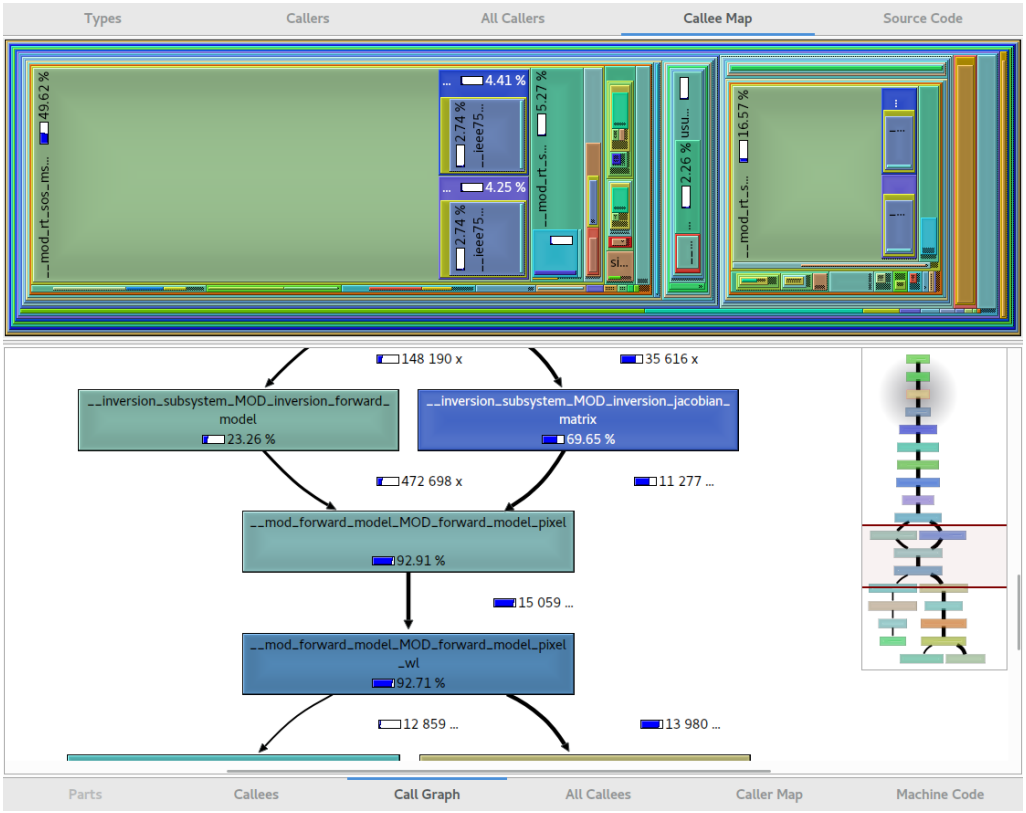
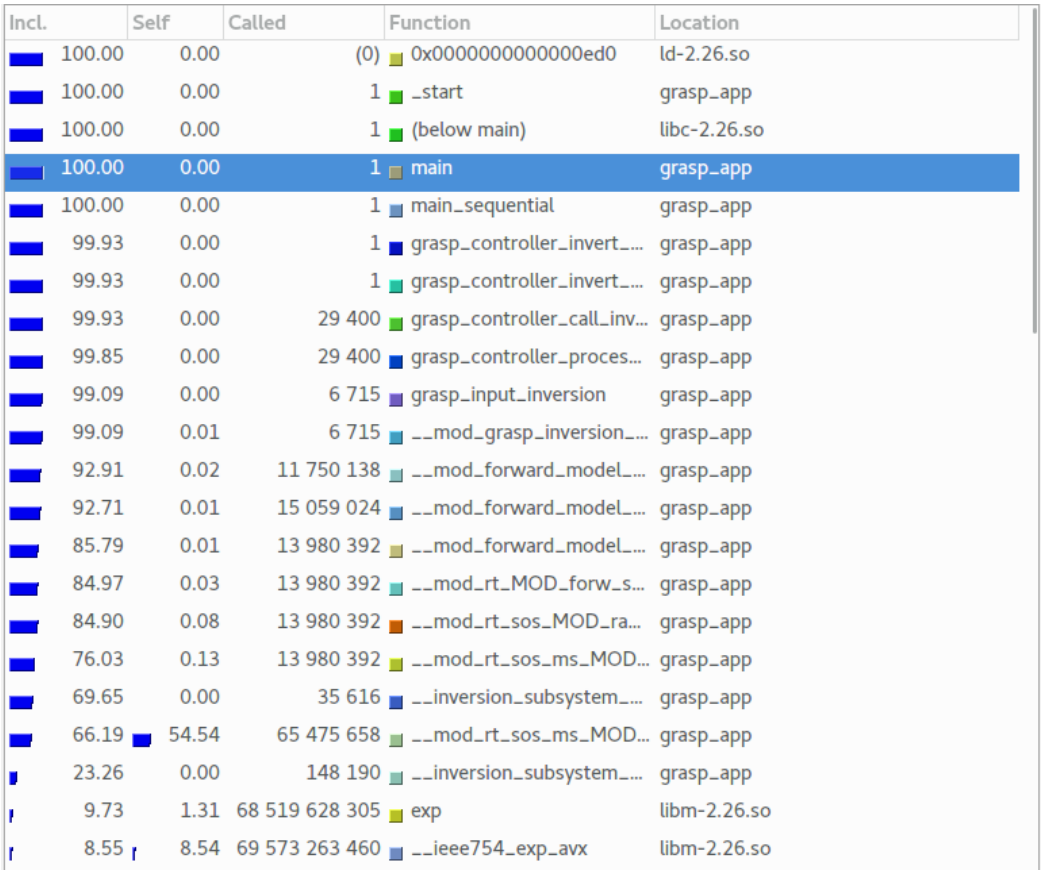


Figure 2: A typical GRASP call-graph, showing the duality of forward model and numerical inversion.

[2] http://valgrind.org/
[3] http://kcachegrind.sourceforge.net/html/Home.html



Incl.	Self	Called	Function	Location
100.00	0.00	(0)	0x0000000000000ed0	ld-2.26.so
100.00	0.00	1	_start	grasp_app
100.00	0.00	1	(below main)	libc-2.26.so
100.00	0.00	1	main	grasp_app
100.00	0.00	1	main_sequential	grasp_app
99.93	0.00	1	grasp_controller_invert_...	grasp_app
99.93	0.00	1	grasp_controller_invert_...	grasp_app
99.93	0.00	29 400	grasp_controller_call_inv...	grasp_app
99.85	0.00	29 400	grasp_controller_proces...	grasp_app
99.09	0.00	6 715	grasp_input_inversion	grasp_app
99.09	0.01	6 715	__mod_grasp_inversion_...	grasp_app
92.91	0.02	11 750 138	__mod_forward_model_...	grasp_app
92.71	0.01	15 059 024	__mod_forward_model_...	grasp_app
85.79	0.01	13 980 392	__mod_forward_model_...	grasp_app
84.97	0.03	13 980 392	__mod_rt_MOD_forw_s...	grasp_app
84.90	0.08	13 980 392	__mod_rt_sos_MOD_ra...	grasp_app
76.03	0.13	13 980 392	__mod_rt_sos_ms_MOD...	grasp_app
69.65	0.00	35 616	__inversion_subsystem_...	grasp_app
66.19	54.54	65 475 658	__mod_rt_sos_ms_MOD...	grasp_app
23.26	0.00	148 190	__inversion_subsystem_...	grasp_app
9.73	1.31	68 519 628 305	exp	libm-2.26.so
8.55	8.54	69 573 263 460	__ieee754_exp_avx	libm-2.26.so

Figure 3: a typical "flat profile", showing inclusive CPU time of a function, and "self time" of a function, with one to two functions standing out as having high values of "self time".

Finding which parts of the code run faster or slower returns a relative result (percent of time spent in a routine), not an absolute result (seconds). Valgrind furthermore causes a slowdown of a factor of 5 - 100, which is why not the whole orbit was processed, but two granules in the Mediterranean over Italy and Libya. These reasons mean it was neither practical nor necessary to use the data of the whole orbit, instead we used two granules per run: one over Libya, a desert area where it's hard to retrieve surface parameters, and one over the Mediterranean, where coastal regions in Italy pose challenges to the convergence of the algorithm.

In total we profiled four different GRASP settings on two different datasets. These eight configurations were easily handled by a shell-script, and didn't necessitate the creation of a scheduling / configuration handling tool.

Having GRASP installation, GRASP settings and benchmark configurations persisted, it was possible to create repeatable benchmarks and profiling runs. The final results of these benchmarks are presented in the next section.

2.5 Impact of Parallax Correction

In a first benchmark we determined that except for test1, which is using a very accurate modelling and thus suffering from an inaccurate correction, the parallax correction doesn't have a significant impact on performance, as evident from Table 2.

Table 2: Time per pixel (in core-sec) for varying parallax corrections.

Parallax Version	min	max	mean	median	5th quantile	95th quantile
test3.sd.mp.models.8wl.fix_surf_noparallax	0,04	2,45	0,62	0,56	0,18	1,3
test3.sd.mp.models.8wl.fix_surf_parallax	0,04	3,13	0,62	0,56	0,18	1,29
test3.sd.mp.models.8wl.fix_surf_parallaxCloud	0,04	3,19	0,62	0,56	0,18	1,3

Consequently, in the following tables, all results are only presented for the "parallaxCloud" dataset, which is generated in the most accurate way and thus the most representative one.

2.6 Time per Pixel

The following Table 3 holds the time per pixel for the 4 configurations:

Table 3: Time per pixel (in core-sec) for varying GRASP retrieval configurations.

Configuration	min	max	mean	median	5th quantile	95th quantile
test1.sd.sp.hp.8wl.full_retrieval_parallaxCloud	0.814	19.063	3.302	3.069	1.628	5.691
test2.sd.mp.opt.8wl.full_retrieval_parallaxCloud	0.053	5.391	0.334	0.350	0.224	0.406
test3.sd.mp.models.8wl.fix_surf_parallaxCloud	0.022	0.819	0.142	0.145	0.078	0.200
test4.sd.mp.models.8wl.apri_surf_parallaxCloud	0.034	1.608	0.234	0.234	0.159	0.301

As noted in the approach above, these numbers cover the full GRASP processing, yet exclude any I/O that might happen at the eventual PDAP facility.

2.7 Impact of Segment Size

All configuration, i.e. test 1-4, have been benchmarked with a varying segment size, from 2x2 up to 25x25. The raw results are provided in Appendix 1, which includes minimum, maximum, average, median and the 5th and 95th percentile of the processing time per pixel. For brevity Table 4 and Table 5 show the average case and worst case (95th percentile) for each tested segment size.

In these tables the GRASP settings (c.f. chapter Repeatability) are shortened from e.g. *test1.sd.sp.hp.8wl.full_retrieval* to *test1*.

Table 4: The average processing time per pixel for each of the tests and different segment sizes.

Segment Size (N x N)	**test1**	**test2**	**test3**	**test4**
2 x 2	3.332694	0.343837	0.16194	0.260974
10 x 10	**3.301864**	**0.334429**	**0.142373**	**0.233828**
15 x 15	3.444605	0.355081	0.14471	0.236726
20 x 20	3.791638	0.381572	0.148211	0.244453
25 x 25	4.422248	0.417411	0.151633	0.255086

Table 5: The worst case processing time per pixel (95th percentile) for each of the tests and different segment sizes.

Segment Size (N x N)	**test1**	**test2**	**test3**	**test4**
2 x 2	6.436282	0.429404	0.222743	0.323748
10 x 10	**5.69118**	**0.406452**	**0.200029**	**0.300891**
15 x 15	5.791542	0.433231	0.201964	0.306818
20 x 20	6.109502	0.485137	0.20483	0.32041
25 x 25	6.792616	0.567345	0.20937	0.340376

Figure 4 and Figure 5 illustrate the test timings for test1-4 and segment size 10x10 which is the fastest of the tested segment sizes.

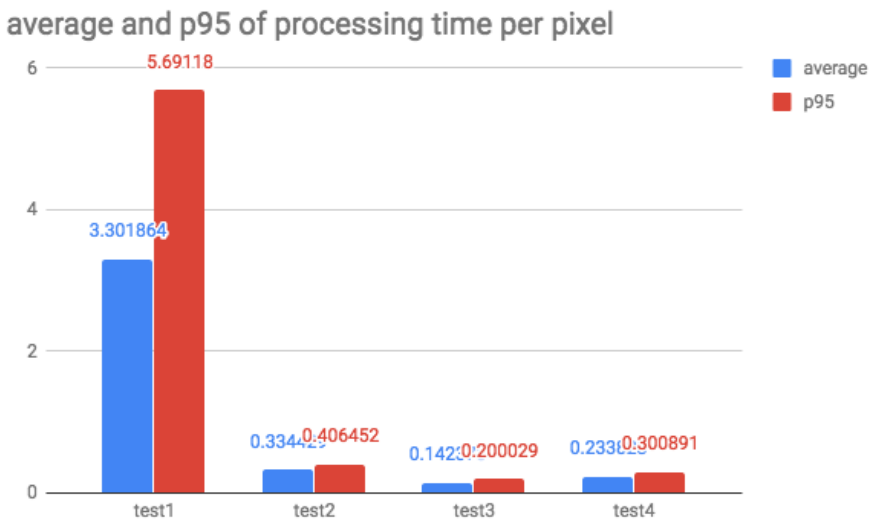


Figure 4: Average time per pixel for GRASP configurations test2 - test4, using segment size 10x10

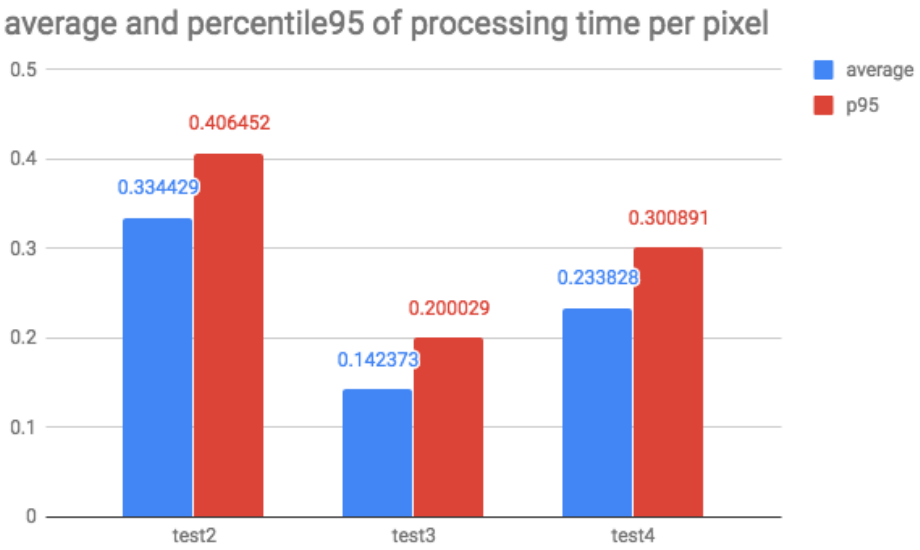


Figure 5: Average time per pixel for GRASP configurations test2 - test4, using segment size 10x10

2.8 Amount of Cores Necessary for NRT

In an effort to estimate the cores necessary for NRT processing, we assumed a 60% cloud coverage and 1.89 million pixels per orbit, as confirmed by the test data set provided. Table 6 contains the core counts that are required to process one such orbit within the 45 minutes of processing time.

Table 6: Core counts that are required to process one orbit with 60% cloud cover in near real-time (45min).

Times and cores for 45 minutes target	Average		5th Quantile (best case)		95th Quantile (worst case)	
Configuration	**core-s / px**	**# cores**	**core-s / px**	**# cores**	**core-s / px**	**# cores**
test 1: full retrieval	3.302	925	1.506	422	5.691	1594
test 2: full retrieval optimized	0.334	94	0.219	62	0.406	114
test 3: fixed surface	0.142	40	0.095	27	0.200	56
test 4: a-priori surface	0.234	66	0.179	51	0.301	85

As further visualized in Figure 6 and zoomed in Figure 7, configuration test 3 and 4 will meet the maximum core-requirement of 100 cores in all cases. Configuration test 2 would meet it in the average case, and with a 10% improvement also in the worst case. Configuration test 1 is not feasible.

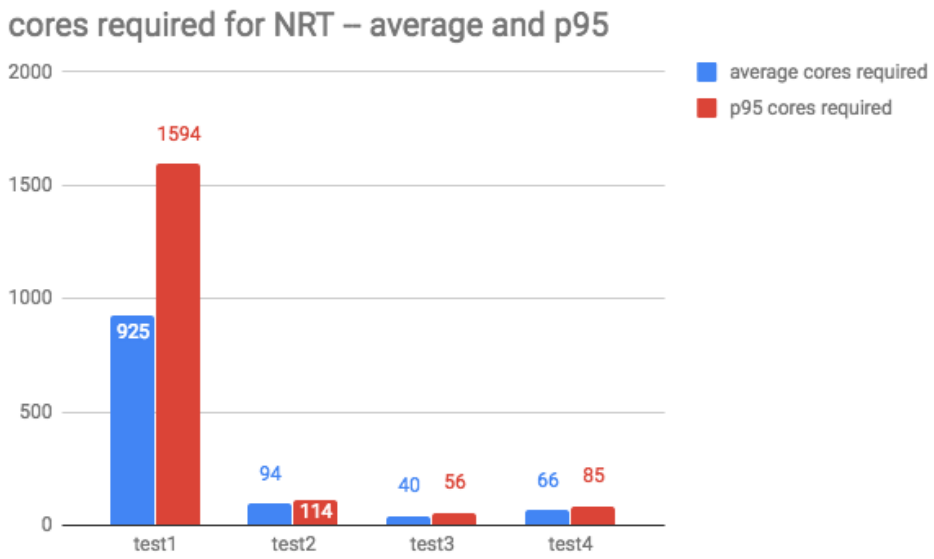


Figure 6: Core counts that are required to process one orbit with different GRASP configurations (all tests)

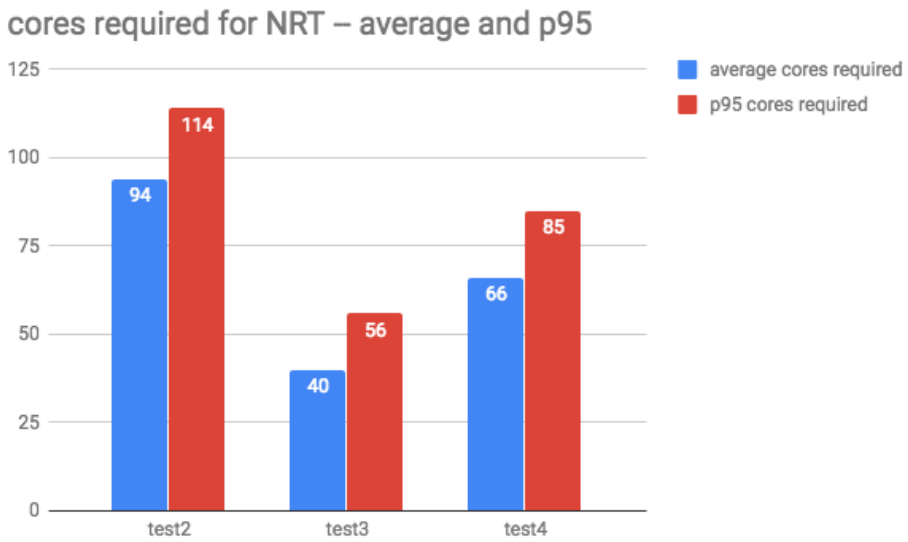


Figure 7: Core counts that are required to process one orbit with different GRASP configurations (only test 2-4)

In 2018, the maximum cores that can be included in a single CPU socket are 22 for Intel Xeon E5, and 32 for AMD Epyc. Consequently it would require 1 quad-socket configuration (equivalent to the current Sentinel-3 processing node) today. Given that the number of cores is the primary scaling factor for CPUs today - as opposed to the GHz race in the 90s and early 2000s - it is possible that by the time of a 3MI launch a single CPU socket will already feature 48 cores, and thus 1 dual-socket node is sufficient for the NRT processing.

2.9 Profiling

This section contains key findings from profiling GRASP with all four configurations on two granules / overlaps over Italy or Libya respectively.

A table with raw results of the profiling is provided in Appendix 2. This includes the two functions in GRASP that the most time was spent in, as well as the total percentage of execution time that is explained by those two functions.

In the following tables the GRASP settings (c.f. chapter Repeatability) are shortened from e.g. *test1.sd.sp.hp.8wl.full_retrieval* to *test1*.

2.9.1 Potential for Optimization

Table 7 contains the total percentage of execution time that is explained by the top two functions that most time is spent in. Higher values mean that optimizing those two functions leads to a higher speed-up of the

program in total, according to Amdahl's law[4]. Low numbers mean that the two "dominant" functions aren't dominant enough to enable substantial speed-up by optimizing them. E.g. when a function is using 30% of the time, no matter how much it is accelerated, the total run time would only decrease by those 30%.

Table 7: Percentage of total processing time that is explained by the two most dominant functions w.r.t. execution time

GRASP configuration	**total % explained**
test1	87.89
test2	69.93
test3	72.16
test4	45.97

This means that assuming we can speed up the two most time-intensive functions by a factor of two, in the configuration *test1* GRASP could be sped up a factor of ~1.8. In configurations test2 and test3 that potential decreases to a speed-up of 1.5 and with test4 it is down to 1.3. This is due to Amdahl's law.

2.9.2 Identification of Most Time-Intensive Functions

Table 8 shows a list of functions that incur the highest percentage of time-spent and second-highest percentage of time spent for each test configuration, and data location.

Table 8: functions with highest percentage of time-spent and second-highest percentage of time spent for each test configuration, and data location.

Configuration, Granule location	**Function 1 (highest percentage of time spent)**	**Execution Time Share**	**Function 2 (second-highest percentage of time spent)**	**Execution Time Share**
test1, Italy and Libya	__mod_rt_sos_ms_MOD_ordn	80.48 %	__ieee754_exp_avx	7.41 %
test2, Italy and Libya	__mod_rt_sos_ms_MOD_ordn	59.66 %	__ieee754_exp_avx	10.28 %

[4] "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", Amdahl, Gene M., 1967; see also https://en.wikipedia.org/wiki/Amdahl%27s_law.

test3, Libya	__mod_rt_sos_ms_MOD_ordn	64.88 %	__ieee754_exp_avx	9.38 %
test3, Italy		61.56 %	__memset_avx2_erms	8.50 %
test4, Italy and Libya	__mod_rt_sos_ms_MOD_ordn	34.13 %	__mod_rt_sos_ms_MOD_phmx_m	11.85 %

In most cases the most time-demanding function (function 1 in the nomenclature of Table 8) was __mod_rt_sos_ms_MOD_ordn, the "Higher Order of Scattering" function. Increasing the performance of this function has the potential to greatly improve the performance of GRASP in all configurations. This function is a highly mathematical one though, which means improving it will most certainly mean modifying the underlying mathematics and using approximations or other tools to speed up the computation. This is harder to achieve than standard performance tuning that doesn't touch the physical background of a numerical code.

The second most time-demanding function (function 2 in the nomenclature of Table 8) was in many cases __ieee754_exp_avx which are AVX CPU instructions for calculating of the exponential function. In configurations test1 - test3 GRASP spent between 7 and 10% of the time calculating exponentials e^x. In test3, over Italy, the second-most impactful function was __memset_avx2_erms which is the AVX2 ERMS instruction for copying memory. This was called mostly from functions __mod_fisher_matrix_ccs_MOD_uf_nonzero and grasp_controller_processor_unit. Potentially some of the copying could be reduced after studying the internals of GRASP, yielding a potential speed-up of at most 4-9%. In test3, the second-most time-consuming function was __mod_rt_sos_ms_MOD_phmx_m, the "Mth Fourier Terms of the Phase Matrix of each Atmosphere Component" which consumed 11.8% of the total processing time of GRASP. This function, if worked on hard, has the potential to speed-up execution times of GRASP with configuration test4 by 6-13%.

Summarizing the function __mod_rt_sos_ms_MOD_ordn has by far the highest impact on GRASP performance, being the function that is responsible for 35-80% of the execution time of GRASP. Following that are already direct CPU instructions like __ieee754_exp_avx and __memset_avx2_erms which cannot be improved by themselves anymore; the only way to decrease the total processing time of instructions like these is by calling them fewer times.

2.10 Conclusion

In summary GRASP meets the performance requirements to serve EUMETSAT in the near real-time calculation of atmospheric and surface properties for the 3MI instrument. The processing time is well within the time limit, assuming a moderate amount of processing cores available.

Specifically the "multi-pixel models approach with fixed surface characteristics" (test3.sd.mp.models.8wl.fix_surf) and the "multi-pixel models approach with a-priori surface" (test4.sd.mp.models.8wl.apri_surf) are well suited to be included in the eventual processing environment. Full retrievals of both atmospheric- and surface-properties with "optimized settings" but without initial guesses for surface parameters (test2.sd.mp.opt.8wl.full_retrieval) are on the slower end of the spectrum of these perfor-

mance tests, but not entirely unfeasible. The retrievals with “optimized settings” and with initial guesses for surface or with fixed surface also can be considered as suitable approach for NRT if aerosol model independent approach and better retrieval accuracy are required.

The benchmark statistics have helped to understand the impact of the various configuration setups and the processing segment sizes. In the tested configurations a 10x10 setup yielded the best results. For the operational processing, it will be necessary to revisit this segment size matter, as it will be a balance of the requested parallelism of PDAP on one side and the performance quality on the other. Larger segment sizes will reduce the amount of segments that can be processed in parallel but it will improve the information for the retrieval.

The amount of cores necessary has been estimated on the single simulated orbit provided by EUMETSAT. The optional activity to validate these numbers by processing one year of PARASOL orbits as proxy was not activated by EUMETSAT.

As for performance improvements, there is one function, `__mod_rt_sos_ms_MOD_ordn`, that stands out, where modifications can potentially improve GRASP performance yet more. The other dominant function is a memory copy instruction, which could be improved by revising the memory handling and initialization.

3 Ground Segment Integration Assessment

3.1 Introduction

All data processors handling and generating data products are integrated in processing facilities. The facility that is destined to handle the operational processing of the 3MI instrument is the Payload Data Acquisition and Processing (PDAP) environment, which is currently being constructed. Given the particular specialties this environment will bring that break with traditional practices, e.g. enforcing the usage of the Java programming language and enforcing the concept of stream processing, several questions arise about a future implementation of GRASP:

- Whether GRASP can be implemented in the PDAP environment at all
- How GRASP can fit into the PDAP environment and necessary modifications to GRASP and PDAP respectively
- Compatibility with and/or necessary adaptations to the 3MI L2 Aerosol Processor Interface for GRASP

3.2 Approach

In order to assess the feasibility of fitting GRASP within the PDAP environment the challenge has been that this environment does not yet exist and is still in development with information about it being scarce. However, based on background knowledge and the fact that the design principles it follows are not novel, the underlying technologies could be evaluated and analyzed independently.

As for assessing the processor interfaces the specification drafted so far has ben provided by EUM and is precise.

3.3 Implementing GRASP in PDAP

Based on presentation provided at the kickoff[5], and information gathered from the comparable MTG L2PF[6] - which is implemented by the same contractor and following the same practices - PDAP makes use of a stream-processing environment on top of Apache Storm[7] and Apache Kafka[8]. The exact mode of operation of PDAP, e.g. how many pixels are to be processed at a time, is still not finalized. Consequently several possible scenarios are considered.

Based on the assessment of the constraints and the capacity of GRASP, three different modes of running GRASP in PDAP are suggested. The validity of all three modes of operation has been confirmed by EUM.

[5] GRASP-3MI-Study_Meeting_2018-04-11_KO-PDAP-constraints.pptx
[6] MTGL2-APE-RUL-044-TS, Version 1.0 - 30/08/2016, Thales Services SAS
[7] http://storm.apache.org/
[8] https://kafka.apache.org/

3.3.1 Single-Pixel Retrieval

The first mode represents a single-pixel retrieval, where no spatial or temporal information is considered in the retrieval. This is the easiest approach to fit within the PDAP ground segment. However, it also uses the least information and its usage might be discouraged from the scientific teams. The workflow is outlined in Figure 8.

The processing facility has full freedom of whether to schedule the processing of individual pixels or granules:

- In case of a single pixel as input, GRASP outputs the retrieved parameters for this single pixel.
- For a granule as input, GRASP iterates over all pixels in this granule, processes each pixel individually and outputs a granule of results.

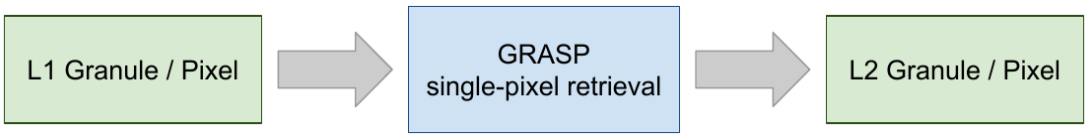


Figure 8: A single-pixel workflow for a GRASP stream processing.

3.3.2 Multi-Pixel Retrieval For Granules

The second mode of operation is a multi-pixel processing using spatial information, but no temporal information. GRASP accepts a granule as input and considers all pixels in the granule during the retrieval to achieve better results. The output is again a granule. This workflow is outlined in Figure 9.

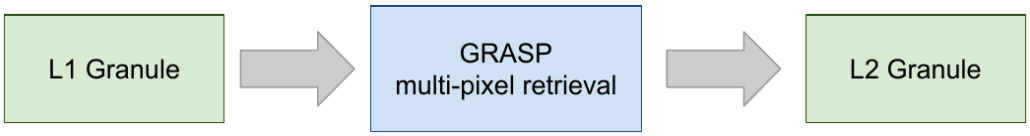


Figure 9: A (spatial) multi-pixel workflow for GRASP stream processing with granules.

This mode of operation requires the availability of a granule holding a certain number of spatially close pixels. The minimal amount of pixels required depends on the configuration, but it is typically a rectangular area of less than 100 by 100 pixels.

3.3.3 Multi-Pixel Retrieval For Single Pixels

The third approach applies a multi-pixel retrieval, like in the mode above, but under the assumptions that only single pixels are being scheduled for processing. Before a multi-pixel retrieval can be applied, the provided pixels need to be collected and aggregated to a segment in a join step. In the example, a segment size of 20x20 pixels is selected. This segment will then be processed into 20x20 output pixels. After the processing, this output segment is split up and the pixels are passed on separately to preserve the level of parallelism of the input. The overall workflow is outlined in Figure 10.

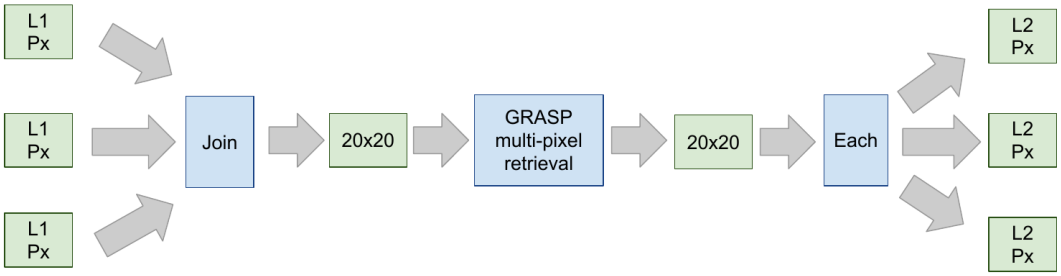


Figure 10: A (spatial) multi-pixel workflow for GRASP stream processing with single-pixels.

3.4 Required Modifications of GRASP or PDAP

No modification to GRASP as algorithm are required for its embedding in PDAP.

When using GRASP for processing single-pixel input in a multi-pixel retrieval, the aggregation of the input and separation of the output need to be implemented. This functionality for this is present within PDAP, and it is a matter of setting up the processing graph to support it.

In the event that some modules of GRASP are to be integrated as customer furnished items (CFI), a wrapper will be required to call the GRASP routines which are likely developed in Fortran or C++. Based on the discussions on the MTG L2PF, this functionality is likely present within PDAP due to other requirements.

As a recommendation, it would be good if PDAP would drop the hard requirement on Java as implementation language, in favor of a recommendation. While both Apache Storm and Kafka promote the usage of Java, they also both offer interfaces for other languages. With the future of Java not as certain as it was several years ago, and the trend going to more expressive languages like Python[9], it would be well advised not to end up in a lock-in situation.

3.5 Compatibility with the Aerosol Processor Interface

The Aerosol Processor Interface as specified in the PGS[10] contains all data required for running GRASP. Figure 11 is showing the annotated figure, giving required, optional, and unnecessary fields.

[9] https://insights.stackoverflow.com/trends?tags=python%2Cjava%2Cc%2B%2B

[10] EUM/LEO-EPSSG/SPE/14/772046, v3A, 6 November 2017, EUMETSAT

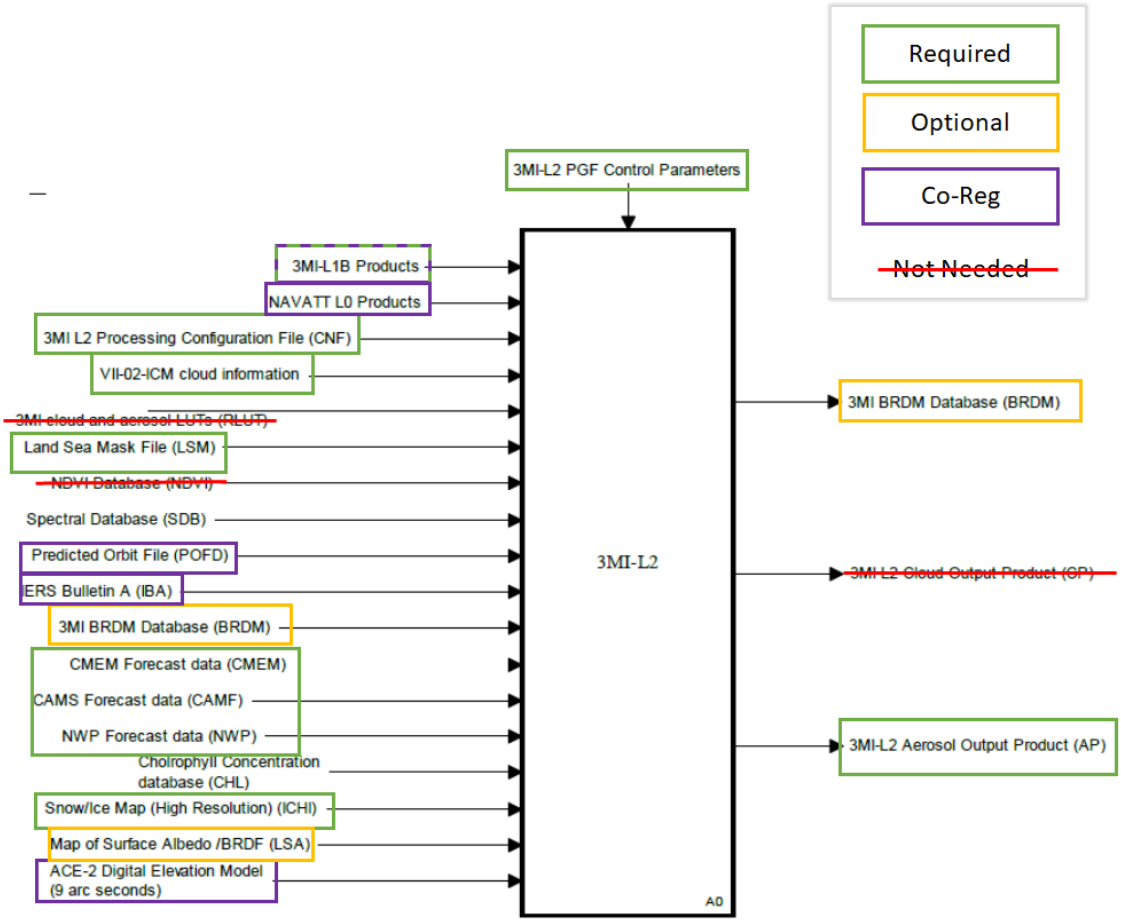


Figure 11: The annotated Figure 15 from the current Aerosol PGS for 3MI.
The upper box holds the legend of the annotations.

The 3MI cloud and aerosol LUTs and the NDVI database are not required by GRASP and could be removed from the processor interface. O_3 and NO_2 are required for forecasts but their availability was already confirmed by EUMETSAT. The BRDM database included in the interface, will be fed by GRASP. The usage of cholorophyll concentration and the spectral database could not be clarified, but are likely not critical.

For this study it was assumed that L1C products are available as input. However, as not all 3MI bands are used, it would continue to make sense to use an embedded L1C processing to save processing wall-time. The data required only for co-registration is marked as such.

3.6 Static Data Tables

Below Table 9 summarizes the static data tables required for running GRASP.

Table 9: The static data tables with their sizes, update cycles, and purpose.

Item	Type	Max Size	Avg Size	Static	Update Cycle	Required	Purpose	Comment
kernels	table	200 MB	150 MB	yes	with new GRASP versions	yes if no models	pre-calculated info for GRASP re-trieval	either kernels or models are always required
models	table	1 MB	1 MB	yes	with new major GRASP versions	yes if no kernels	pre-calculated info for GRASP re-trieval	only used in case we use models settings for GRASP (enhanced speed)
surface a-priori	climatology	equal to output	equal to output	no	bi-weekly	optional	quality en-hancement	if we use a-priori knowledge
surface climatology	climatology	125 MB / overlap	75 MB / over-lap	no	monthly	optional	quality en-hancement	if we use multi-temporal approach

Based on the feedback from the EUM ground segment team, the size and update cycles are confirmed to fit within PDAP.

3.7 Conclusion

Based on the investigation of the available documentation, it is concluded that GRASP can be integrated into the PDAP environment. Neither the GRASP algorithm, nor PDAP needs to be modified. Several modes of operation have been drafted to give the science team the necessary flexibility to come up with the optimal retrieval settings.

No extension of the Aerosol Processor Interface are required, all relevant data is present. Some input data can be removed.

4 Documentation Assessment

4.1 Introduction

Integrating GRASP in the overall 3MI frameworks requires following a strict development plan imposed by the overall EPS-SG program. This includes that all algorithms are specified in sufficient detail to be entirely re-implemented independently in the target processing facility. Due to GRASP's size and complexity compared to classic algorithms, the feasibility of fully specifying the algorithm is to be proven. Code metrics provide an objective base for extrapolating the required specification effort.

The purpose of this document is to:

- Compare different GRASP configurations with respect to their code quantity
- Compare different GRASP configurations with respect to their parameter quantity
- Compare different GRASP configurations with respect to their code complexity
- Extrapolate the effort of creating a complete GRASP specification

4.2 Approach

The high-level approach to estimate the effort needed to specify and document GRASP in the form of a Product Generation Specification (PGS) was following the following steps:

1. Tailor GRASP to the minimum necessary for the target configuration
2. Understand the quantity and complexity of GRASP for the target configuration
3. Select a reference routine, specify it, and not the effort necessary
4. Extrapolate the effort for the reference routine based on the quantity and complexity of the overall setup

For this analysis, four different configurations of GRASP are considered, which are described in Chapter 1.3 in detail:

- **test1.sd.sp.hp.8wl.full_retrieval:** single-pixel, high-precision, full retrieval
- **test2.sd.mp.opt.8wl.full_retrieval:** multi-pixel, optimized, full retrieval
- **test3.sd.mp.models.8wl.fix_surf:** multi-pixel, models, a-priori surface
- **test4.sd.mp.models.8wl.apri_surf:** multi-pixel, models, fixed surface

All configurations use the full vector radiative transfer, and while some kernels are being used, all calculations are still being done online without the use of look-up-tables.

SciTools Understand[11] was used to conduct a static code analysis and generate the presented metrics. Functions that were actually executed in the respective test cases were dynamically identified with Gcov[12]. Note

[11] https://scitools.com/

that unused functions were excluded when generating the metrics, but unused branches within executed functions were still considered. Additionally, the metrics are restricted to the retrieval code of GRASP, and excluded all I/O interfaces, tests, and command line handling.

The input orbit used to run GRASP in the different configurations was provided by EUM:

- W_xx-eumetsat-darmstadt,SAT,SGA1-3MI-1C-REF_C_EUMT_xxxxxxxxxxxxxx_G_D_20080223085157_20080223085241_T_N____.nc

In order to extrapolate the effort of creating a specification, the function "ORDN" was selected as a reference and was documented. The resulting documentation was approved by EUMETSAT to be in an acceptable format for a PGS. The documented "ORDN" function can be found in the Annex.

4.3 Code Quantity

The first metric to be inspected is the overall code quantity of the executed functions. First, the total number of functions is given. Additionally, the total number of lines as well as the number of executable lines of the functions are calculated.

Table 10: The lines of GRASP code for the varying configurations.

Configuration	Functions	Total Lines	Executable Lines
test1.sd.sp.hp.8wl.full_retrieval	165	17392	8875
test2.sd.mp.opt.8wl.full_retrieval	172	18613	9650
test3.sd.mp.models.8wl.fix_surf	158	16703	8738
test4.sd.mp.models.8wl.apri_surf	162	16854	8763

Considering the above metrics, configuration test3.sd.mp.models.8wl.fix_surf has the smallest code base of the tested variants.

The tailoring of GRASP has been limited to the removal of unused functions. An additional removal of unused branches of functions within the execution flow can further lower the overall amount of code. This was not exercised as part of this study as it should only be done in conjunction with a thorough inspection of the code.

[12] https://gcc.gnu.org/onlinedocs/gcc/Gcov.html

4.4 Parameter Quantity

Another important aspect of functions is their data flow. Measured by the sum of the input and output parameters of all their functions, the numbers for all configurations are given below.

Table 11: The number of parameters for the varying configurations.

Configuration	Input Parameters	Output Parameters
test1.sd.sp.hp.8wl.full_retrieval	2006	1033
test2.sd.mp.opt.8wl.full_retrieval	2097	1091
test3.sd.mp.models.8wl.fix_surf	1825	994
test4.sd.mp.models.8wl.apri_surf	1849	1004

Similar to the overall code quantity, test3.sd.mp.models.8wl.fix_surf contained the least total amount of input and output parameters

4.5 Code Complexity

The cyclomatic complexity[13] was chosen as the primary complexity metric in this evaluation. To get an overall impression, the cyclomatic complexity of all functions and subroutines was summed up.

Table 12: The cyclomatic complexity for the varying configurations.

Configuration	Cumulative Cyclomatic Complexity
test1.sd.sp.hp.8wl.full_retrieval	2051
test2.sd.mp.opt.8wl.full_retrieval	2245

[13] Cyclomatic complexity is a software metric indicating the complexity of a program, and was proposed in "A Complexity Measure", Thomas McCabe, 1976, IEEE Transactions on Software Engineering: 308–320. doi:10.1109/tse.1976.233837

test3.sd.mp.models.8wl.fix_surf	2035
test4.sd.mp.models.8wl.apri_surf	2043

As a lower value means less complexity in the code, the third configuration again yielded the best result.

4.6 Extrapolated Documentation Effort

Documentation of "ORDN" showed that 35 input parameters result in 3 pages of documentation. Equally, some 400 lines of code with 300 lines of executable statements yield a documentation with a length of 3 pages. The overall time it took to prepare it was 5-6 man days, but this time range is expected to decrease once the team is able to establish a clear process for the specification.

Function	Parameters	Lines	Executable	Complexity	Pages	Effort
ORDN	35	400	300	57	3	5-6 man days

The effort for is split into four parts:

1. Understanding the routine at hand
2. Simplifying the routine by discard unnecessary options or implementation details
3. Writing the actual specification
4. Independently review the specification to ensure its validity

For an expert staff, with a deep understanding of GRASP, part 1) is neglectable. Still, it should be considered in any estimation as a change in staff can lead to a loss of expertise, and hence poses a risk. Part 2) will be in most cases the most time-consuming. Part 3) is mostly a doing thing, which ideally is aided by machines, e.g. via automated code generation. Part 4) is important to ensure that the specification that has been written is valid. If it was generated automatically from a working, validated implementation this review will be less critical than for handwritten specifications.

Extrapolating from the metrics generated by documenting the function "ORDN", it shows that a complete specification of GRASP would approximately yield:

- 300 pages of input parameter description,
- 300-600 pages of code description,
- 300-500 pages of introduction and general explanations.

Roughly speaking, a total number of 1000-1500 pages of specification can be expected. It shall be noted, that there is still a 30% uncertainty attached to these numbers, however in both directions.

Within the retrieval packages, there are also routines to handle internal/external interfaces (e.g. SuperLU), wrappers (e.g. phasekernel for spheroid package, and forward_model wrapper), and copying data from one structure to another (e.g. sdata). Given that this is an implementation detail and not relevant for the core algorithm, a significant reduction can be achieved. It is estimated that about 400-500 pages could be removed additionally from the above numbers.

Another optimization of code quantity can be achieved by providing complex code sections as a commercial-off-the-shelf (COTS) module. The specification work is then restricted to the interface of the module, and the high level description thereof. The additional benefit would be, that the module could be thoroughly tested and efficiently implemented. Both are things that are not guaranteed by an independent third-party implementation.

Candidates for the integration as COTS modules would be:

- Spheroid Package
- Forward Model

For the extrapolation of effort, an unmodified version of GRASP with a specification length of 1000-1500 pages is assumed. The "ORDN" specification has a length of 6 pages and can realistically be created within 4 man days. Thus 650-1000 man days (3-4.75 man years) are estimated for writing the specification of the overall code. This includes both the effort for simplification and preparing the actual document.

Assuming a 3MI launch date of 2021-2022, this leaves approx. 4 years for the specification and eventual implementation. Considering a 1:3 ratio between specification and implementation - to accommodate prototype and operational processor developments, but excluding the time for contract setups, 1-1.5 years are left to complete the specification. This would require an average team size of 2-3 personnel, with a peak of 4-5 in the early phases.

4.7 Conclusion

All of the provided metrics suggest that GRASP configuration test3.sd.mp.models.8wl.fix_surf results in the least documentation effort. However, it is worth noting that for all metrics the other scenarios differ from the best one by not more than 13%.

The estimation effort is considered realistic, as is the staffing situation at the contractors. Nonetheless, the document will be very large, and it requires extra diligence to ensure that no mistakes end up in the specification. Given that it also requires even more diligence to ensure that no mistakes end up in the eventual implementation, a more incremental and interwoven approach is recommended.

5 Conclusion

This study has assessed GRASP in three distinctive regards:

1. It has benchmarked the computational performance of GRASP wrt. NRT requirements,
2. it has analyzed the possibilities of integrating GRASP in the PDAP ground segment, and
3. it has evaluated the feasibility of specifying GRASP in a detailed specification.

All three assessments have a positive outcome, and, therefore, no major blockers are to be expected for an operational integration of GRASP for EPS-SG/3MI from either of these three engineering perspectives.

The detailed conclusions for each perspective can be found in Chapter 2.10, Chapter 3.7, and Chapter 4.7. From the overall perspective, specifically the GRASP setup with fixed surface information or an initial guess for it – which can and should be taken from GRASP itself – show a good performance fitting within the NRT requirements, and the least amount of documentation effort.

It shall be noted again that, although considered feasible, the overall effort and susceptibility to errors of the detailed-specification approach to implementing processors in general is a risk and it is recommended to be replaced in favor with a more incremental way of realizing the final data processors, which includes the key benefits of the traditional approach: a through scientific and engineering review of the implementation along with a detailed specification of the product being generated.

Appendix 1 – Detailed Performance results per segment size

	min	max	mean	median	pct 5	pct 95	n_px_loaded	frac_processed	n_px_ignored	total_time_s	algo_time_s	finished
2x2_test1.sd.sp.hp.8wl.full_retrieval	0.387	23.415	3.333	2.995	1.477	6.436	1889342	0.273254	927	1723657	1720572	TRUE
2x2_test2.sd.mp.opt.8wl.full_retrieval	0.054	5.442	0.344	0.361	0.215	0.429	1889342	0.273254	927	181292.3	177513.4	TRUE
2x2_test3.sd.mp.models.8wl.fix_surf	0.025	0.855	0.162	0.166	0.092	0.223	1889342	0.273207	1015	88478.99	83591.88	TRUE
2x2_test4.sd.mp.models.8wl.apri_surf	0.036	2.217	0.261	0.260	0.176	0.324	1889342	0.273207	1015	138918.2	134711.6	TRUE
10x10_test1.sd.sp.hp.8wl.full_retrieval	0.814	19.063	3.302	3.069	1.628	5.691	1889342	0.268448	10006	1675164	1674676	TRUE
10x10_test2.sd.mp.opt.8wl.full_retrieval	0.053	5.391	0.334	0.350	0.224	0.406	1889342	0.268448	10006	170109.7	169620.1	TRUE
10x10_test3.sd.mp.models.8wl.fix_surf	0.022	0.819	0.142	0.145	0.078	0.200	1889342	0.267953	10943	72569.96	72077.34	TRUE
10x10_test4.sd.mp.models.8wl.apri_surf	0.034	1.608	0.234	0.234	0.159	0.301	1889342	0.267953	10943	118850	118376.7	TRUE
15x15_test1.sd.sp.hp.8wl.full_retrieval	0.981	21.834	3.445	3.230	1.744	5.792	1889342	0.263874	18648	1.72E+06	1.72E+06	TRUE
15x15_test2.sd.mp.opt.8wl.full_retrieval	0.114	5.417	0.355	0.368	0.239	0.433	1889342	0.263874	18648	1.78E+05	1.77E+05	TRUE
15x15_test3.sd.mp.models.8wl.fix_surf	0.022	0.967	0.145	0.147	0.077	0.202	1889342	0.263124	20065	7.25E+04	7.19E+04	TRUE
15x15_test4.sd.mp.models.8wl.apri_surf	0.034	1.632	0.237	0.237	0.159	0.307	1889342	0.263124	20065	1.18E+05	1.18E+05	TRUE
20x20_test1.sd.sp.hp.8wl.full_retrieval	0.999	18.255	3.792	3.582	1.966	6.110	1889342	0.259254	27378	1.86E+06	1.86E+06	TRUE
20x20_test2.sd.mp.opt.8wl.full_retrieval	0.080	4.534	0.382	0.392	0.250	0.485	1889342	0.259254	27378	1.87E+05	1.87E+05	TRUE
20x20_test3.sd.mp.models.8wl.fix_surf	0.044	1.111	0.148	0.150	0.079	0.205	1889342	0.257942	29856	7.28E+04	7.22E+04	TRUE
20x20_test4.sd.mp.models.8wl.apri_surf	0.039	1.890	0.244	0.241	0.159	0.320	1889342	0.257942	29856	1.20E+05	1.19E+05	TRUE
25x25_test1.sd.sp.hp.8wl.full_retrieval	1.837	20.804	4.422	4.300	2.413	6.793	1889342	0.254248	36836	2.12E+06	2.12E+06	TRUE

	min	max	mean	median	pct 5	pct 95	n_px_loaded	frac_processed	n_px_ignored	total_time_s	algo_time_s	finished
25x25_test2.sd.mp.opt.8wl.full_retrieval	0.171	5.131	0.417	0.423	0.279	0.567	1889342	0.254248	36836	2.01E+05	2.01E+05	TRUE
25x25_test3.sd.mp.models.8wl.fix_surf	0.044	1.786	0.152	0.153	0.084	0.209	1889342	0.252682	39795	7.30E+04	7.24E+04	TRUE
25x25_test4.sd.mp.models.8wl.apri_surf	0.060	2.587	0.255	0.249	0.169	0.340	1889342	0.252682	39795	1.22E+05	1.22E+05	TRUE

Appendix 2 – Detailed Profiling Results

Test	total % ex-plained	func1	func1 % self	func1 main (transitive) callers and their inclusive %	func2	func2 % self	func2 main (transitive) callers and their inclusive %
test1, Italy	86.22	__mod_rt_sos_ms_MOD_ordn	78.19	__inversion_subsystem_MOD_inversion_forward_model (31.73%) __inversion_subsystem_MOD_inversion_jacobian_matrix (67.22%)	__ieee754_exp_avx	8.03	__mod_rt_sos_ms_MOD_int1 (0.60%) __mod_rt_sos_ms_MOD_intn (3.63%) __mod_rt_sos_ms_MOD_int1t2 (3.63%)
test1, Libya	89.56	__mod_rt_sos_ms_MOD_ordn	82.77	__inversion_subsystem_MOD_inversion_forward_model (31.73%) __inversion_subsystem_MOD_inversion_jacobian_matrix (67.22%)	__ieee754_exp_avx	6.79	__mod_rt_sos_ms_MOD_int1 (0.34%) __mod_rt_sos_ms_MOD_intn (3.19%) __mod_rt_sos_ms_MOD_int1t2 (3.19%)
test2, Italy	64.27	__mod_rt_sos_ms_MOD_ordn	54.54	__inversion_subsystem_MOD_inversion_forward_model (23.26%) __inversion_subsystem_MOD_inversion_jacobian_matrix (69.65%)	__ieee754_exp_avx	9.73	__mod_rt_sos_ms_MOD_int1 (0.62%) __mod_rt_sos_ms_MOD_intn (4.22%) __mod_rt_sos_ms_MOD_int1t2 (4.22%)
test2, Libya	75.59	__mod_rt_sos_ms_MOD_ordn	64.77	__inversion_subsystem_MOD_inversion_forward_model (24.06%) __inversion_subsystem_MOD_inversion_jacobian_matrix (71.92%)	__ieee754_exp_avx	10.82	__mod_rt_sos_ms_MOD_intn (4.97%) __mod_rt_sos_ms_MOD_int1t2 (4.97%)
test3, Italy	70.06	__mod_rt_sos_ms_MOD_ordn	61.56	__inversion_subsystem_MOD_inversion_forward_model (36.27%) __inversion_subsystem_MOD_inversion_jacobian_matrix (54.75%)	__memset_avx2_erms	8.5	__mod_fisher_matrix_ccs_MOD_uf_nonzero (6.04%) grasp_controller_processor_unit (1.91%)
test3, Libya	74.26	__mod_rt_sos_ms_MOD_ordn	64.88	__inversion_subsystem_MOD_inversion_forward_model (35.93%) __inversion_subsystem_MOD_inversion_jacobian_matrix (59.49%)	__ieee754_exp_avx	9.38	__mod_rt_sos_ms_MOD_int1 (0.60%) __mod_rt_sos_ms_MOD_intn (4.87%) __mod_rt_sos_ms_MOD_int1t2 (4.87%)
test4, Italy	46.31	__mod_rt_sos_ms_MOD_ordn	34.21	__inversion_subsystem_MOD_inversion_forward_model (27.24%) __inversion_subsystem_MOD_inversion_jacobian_matrix (59.16%)	__mod_rt_sos_ms_MOD_phmx_m	12.1	__mod_rt_sos_ms_MOD_rt_sos_ms (61.12%)
test4, Libya	45.63	__mod_rt_sos_ms_MOD_ordn	34.04	__inversion_subsystem_MOD_inversion_forward_model (26.46%) __inversion_subsystem_MOD_inversion_jacobian_matrix (60.59%)	__mod_rt_sos_ms_MOD_phmx_m	11.59	__mod_rt_sos_ms_MOD_rt_sos_ms (59.76%)

Appendix 3 ORDN Specification

Subroutine "ORDN"

Complexity: Medium

Generic Sub-functions

None

Auxiliary Sub-functions

Integrals on optical thickness integration (INT1T2), Stokes vector calculation in a layer (INTN)

Objective of the Function

The subroutine calculates normalized Stokes parameters I,Q,U for given single scattering properties and aerosol optical depth (AOD) of the atmosphere.

Description of the Function Variables

Variables.

Table1. Variables used for ORDN

Symbol	*Descriptive Name*	*Type*	*Unit*	*I/O*	*Source/ Destination*	*References/ Remarks*
m	Number of component of Fourier decomposition	d	-	I	RT_SOS_MS	
l	Index of height level where results will be calculated	d	-	I	RT_SOS_MS	
N_l	Number of vertical levels	d	-	I	RT_SOS_MS	
N_{mx}	Number of atmosphere components	d	-	I	RT_SOS_MS	
WD	Relative atmosphere components loading in each layer	d(N_l-1, N_{mx})	-	I	RT_SOS_MS	
NG	Number of atmosphere components	d		I	RT_SOS_MS	
UG	Gaussian quadrature points	d(-NG:NG)		I	RT_SOS_MS	
WG	Weights of Gaussian quadratures	d(-NG:NG)		I	RT_SOS_MS	
EPS	Required precision of RT calculations	d	-	I	RT_SOS_MS	

P11	Fourier decomposition of element 11 of the phase matrix of atmosphere components	d(-NG:NG, NG,NMX)	-	I	RT_SOS_MS	
P21	Fourier decomposition of element 21 of the phase matrix of atmosphere components	d(-NG:NG, NG,NMX)	-	I	RT_SOS_MS	
P31	Fourier decomposition of element 31 of the phase matrix of atmosphere components	d(-NG:NG, NG,NMX)	-	I	RT_SOS_MS	
P32	Fourier decomposition of element 32 of the phase matrix of atmosphere components	d(-NG:NG, NG,NMX)	-	I	RT_SOS_MS	
P33	Fourier decomposition of element 33 of the phase matrix of atmosphere components	d(-NG:NG, NG,NMX)	-	I	RT_SOS_MS	
R11	Fourier decomposition of element 11 of surface BRM	(NG,NG)	-	I	RT_SOS_MS	
R21	Fourier decomposition of element 21 of surface BRM	(NG,NG)	-	I	RT_SOS_MS	
R22	Fourier decomposition of element 22 of surface BRM	(NG,NG)	-	I	RT_SOS_MS	
R31	Fourier decomposition of element 31 of surface BRM	(NG,NG)	-	I	RT_SOS_MS	
R32	Fourier decomposition of element 32 of surface BRM	(NG,NG)	-	I	RT_SOS_MS	
R33	Fourier decomposition of element 33 of surface BRM	(NG,NG)	-	I	RT_SOS_MS	
YI1	Differences of normalized Stokes parameter I on two levels	d	-	ORDN	internal	
YQ1	Differences of normalized Stokes parameter Q on two levels	d	-	ORDN	internal	
YU1	Differences of normalized Stokes parameter U on two levels	d	-	ORDN	internal	
YI	Integration over optical depth of normalized Stokes parameter I in each level	d	-	ORDN	internal	
YQ	Integration over optical depth of normalized Stokes parameter Q in each level	d	-	ORDN	internal	
YU	Integration over optical depth of normalized Stokes parameter U in each level	d	-	ORDN	internal	
X	Source function of normalized	d	-	ORDN	internal	

	Stokes parameter I in each layer					
Y	Source function of normalized Stokes parameter Q in each layer	d	-	ORDN	internal	
Z	Source function of normalized Stokes parameter U in each layer	d	-	ORDN	internal	
SFI	Source function of normalized Stokes parameter I in each layer and for each angle	d(-NG:NG, NL-1)	-	ORDN	internal	
SFQ	Source function of normalized Stokes parameter Q in each layer and for each angle	d(-NG:NG, NL-1)	-	ORDN	internal	
SFU	Source function of normalized Stokes parameter U in each layer and for each angle	d(-NG:NG, NL-1)	-	ORDN	internal	
Zmax	Maximal value	d	-	ORDN	internal	
I1	Fourier decomposition of the normalized Stokes parameter I on each level	(-NG:NG,NL)	-	I/O	RT_SOS_MS/ORDN	
Q1	Fourier decomposition of the normalized Stokes parameter Q on each level	(-NG:NG,NL)	-	I/O	RT_SOS_MS/ORDN	
U1	Fourier decomposition of the normalized Stokes parameter U on each level	(-NG:NG,NL)	-	I/O	RT_SOS_MS/ORDN	
IM	Multiple order of scattering contribution to Fourier decomposition of the normalized Stokes parameter I	(-NG:NG)	-	O	ORDN	
QM	Multiple order of scattering contribution to Fourier decomposition of the normalized Stokes parameter Q	(-NG:NG)	-	O	ORDN	
UM	Multiple order of scattering contribution to Fourier decomposition of the normalized Stokes parameter U	(-NG:NG)	-	O	ORDN	

Calculate the surface reflection matrix elements using reciprocity principle

$R_{ij}^{nk} = R_{ji}^{kn} / UG_n \text{´ } UG_k$, where i=1..3, j=(i+1)..3, n=1..NG, k=1..NG

Initialize N=1

Initialize $IM_{j=1..NG} = 0$
Initialize $QM_{j=1..NG} = 0$
Initialize $UM_{j=1..NG} = 0$

loop 1: over order of scattering N

loop 2: for vertical levels L from 1 to N_f-1

loop 3: for Gaussian quadrature angles UG_j from j=-NG to NG (when j ¹ 0)

Initialize X=0
Initialize Y=0
Initialize Z=0

Step 1: calculate integrals over optical depth INTL1, INTL2
calling subroutine **INT1T2**

Step 2: calculating source function performing integration over angle

loop 4: for Gaussian quadrature angles UG_k from k = - NG to NG (when k ¹ 0)

calculate

$$YI_{kL}^{I} = (I_{kl+1}^{(1)} + I_{kl}^{(1)}) \text{´ } INTL_1 + \frac{1}{2} INTL_2 \text{´ } (I_{kl+1}^{(1)} - I_{kl}^{(1)})$$

$$YQ_{kl}^{I} = (Q_{kl+1}^{(1)} + Q_{kl}^{(1)}) \text{´ } INTL_1 + \frac{1}{2} INTL_2 \text{´ } (Q_{kl+1}^{(1)} - Q_{kl}^{(1)})$$

$$YU_{kl}^{I} = (U_{kl+1}^{(1)} + U_{kl}^{(1)}) \text{´ } INTL_1 + \frac{1}{2} INTL_2 \text{´ } (U_{kl+1}^{(1)} - U_{kl}^{(1)})$$

loop 5: from n=1 to N_{mx} calculate atmosphere components mixing

$$X = X + (P_{jkn}^{11} YI + P_{jkn}^{12} YQ + P_{jkn}^{13} YQ) \text{´ } WD_{ln} \text{´ } WG_k$$

$$Y = Y + (P_{jkn}^{21} YI + P_{jkn}^{22} YQ + P_{jkn}^{23} YQ) \text{´ } WD_{ln} \text{´ } WG_k$$

$$Z = Z + (P_{jkn}^{31} YI + P_{jkn}^{32} YQ + P_{jkn}^{33} YQ) \text{´ } WD_{ln} \text{´ } WG_k$$

end loop 5

end loop 4

SFI_{jl} = X/2
SFQ_{jl} = Y/2
SFU_{jl} = Z/2

end loop 3

end loop 2

Calculating normalized Stokes vector at ground level (N_l):

loop 2: from j=1 to NG

$$I1_{j,NL} = \sum_{k=1}^{NG} WG_k \cdot \left(R11_{jk} I1_{-k,NL} + R12_{jk} Q1_{-k,NL} + R31_{jk} U1_{-k,NL} \right)$$

$$Q1_{j,NL} = \sum_{k=1}^{NG} WG_k \cdot \left(R21_{jk} I1_{-k,NL} + R22_{jk} Q1_{-k,NL} + R23_{jk} U1_{-k,NL} \right)$$

$$U1_{j,NL} = \sum_{k=1}^{NG} WG_k \cdot \left(R31_{jk} I1_{-k,NL} + R32_{jk} Q1_{-k,NL} + R33_{jk} U1_{-k,NL} \right)$$

end loop 2

Calculating normalized Stokes parameters at each level above the ground level:

Step 1: Down-welling radiation

loop 2: for vertical levels from l=N_l-1 to 1

loop 3: for Gaussian quadrature angles UG_j from j= 1 to NG

Calculate normalized Stokes vector $I1_{jl}, Q1_{jl} U1_{jl}$ at each level *l* and at each Gaussian quadrature points *j* calling subroutine **INTN**

end loop 3

end loop 2

Step 2: Upwelling radiation

loop 2: for Gaussian quadrature angles UG_j from j=-NG to 1

Initialize $I1_{j1} = 0$
Initialize $Q1_{j1} = 0$
Initialize $U1_{j1} = 0$

end loop 2

loop 2: for vertical levels L from 2 to NL

loop 3: for Gaussian quadrature angles UG_j from j=-NG to -1

Calculate normalized Stokes vector $I1_{kl}, Q1_{kl}\ U1_{kl}$ at each level *l* and at each Gaussian quadrature points *j* calling subroutine **INTN**

end loop 3

end loop 2

Calculating order of scattering for normalized Stokes vector at level *IL*

loop 2: for Gaussian quadrature points from j=-NG to NG (when j $\neq$ 0)

$$IM_j = IM_j + I1_{j,IL}$$
$$IM_j = IM_j + I1_{j,IL}$$
$$IM_j = IM_j + I1_{j,IL}$$

end loop 2

Calculating maximum absolute value of Stokes vector (Zmax).

Case 1: If Zmax < EPS exit from the loop 1.

End loop 1