Algorithm Development Library (ADL)

July 20, 2010

ADL Vision

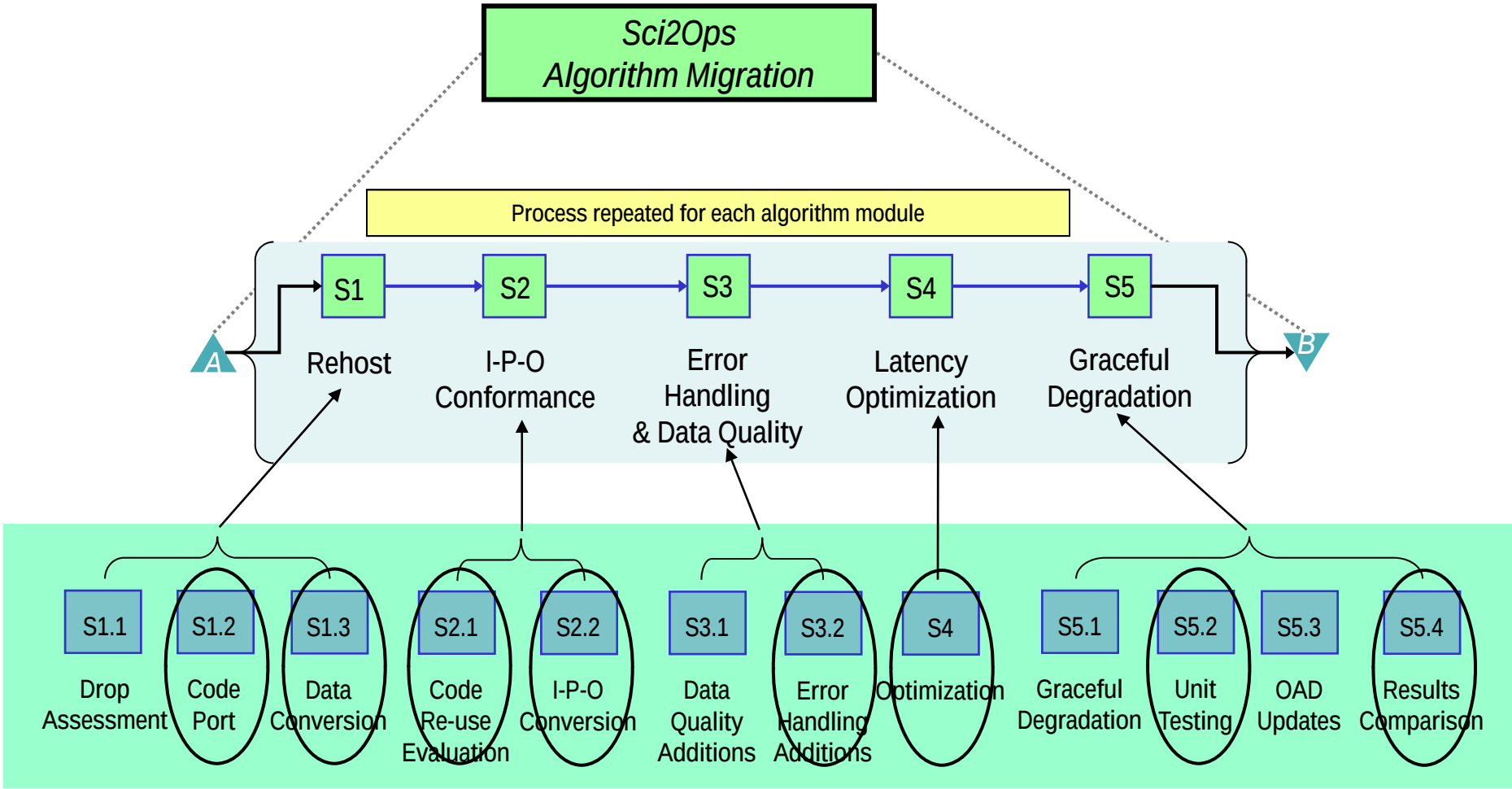
- Provide a tool with a standardized input/output framework, based on operational code and interfaces, into which algorithm scientists can “plug” their algorithms, in their development environment, to develop and debug the algorithms
  - The tool provides Input (“I”) and Output (“O”) interfaces for an algorithm under development that are identical to those of an operationalized algorithm
- Use will significantly decrease the time spent for science-to-operations (Sci2Ops) code conversion
- Will enable return of the operational algorithm to the science algorithm developer who can then insert it back into their ADL environment for further enhancements
  - Code changes to algorithm processing code that are for operational aspects are made to a delegate class to segregate the ops changes to allow for plug and play
  - Allows developer to always work to latest operational baseline
- Increased efficiency for new algorithm developers
  - Auto-generation of code when developing a new algorithm

ADL Background

- Science algorithm to operational algorithm conversion (Sci2Ops) was difficult and therefore costly
  - Need for better coordination and process identified during multiple algorithm Sci2Ops conversions
    - Science algorithms don't always follow any specific execution model, i.e intermixed input, processing, and output
    - No consistent formats or coding standards across algorithm developers
    - Widely varying algorithm designs
    - Widely varying input and output product implementations

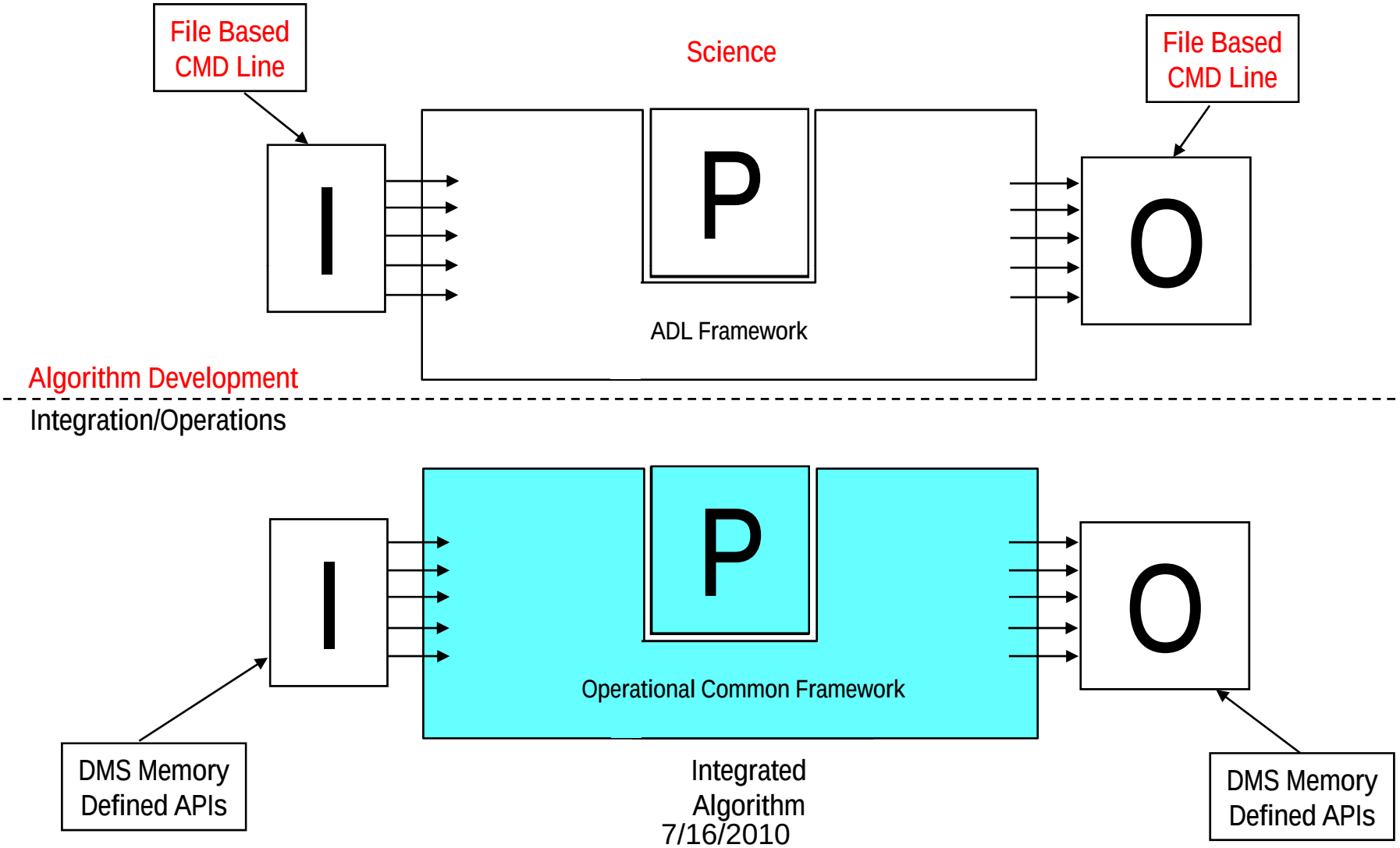
- ADL Design is based on I-P-O (Input-Processing-Output) Model
  - I (Input): Retrieval of Predefined Input Data
    - Previously Created Data Products – RDRs, SDRs, EDRs
    - Ancillary Data – Spacecraft telemetry, climatology data, etc.
    - Auxiliary Data – Look Up Tables (LUTs), Configuration Guides, etc.
  - P (Processing): Processing of Data to Create a Defined Data Product
  - O (Output): Storage of Defined Output Product(s) for Later Retrieval, Formatting, and Delivery

Science-to-Operations Algorithm Migration



Circled steps denote areas of greatest savings

Algorithm Development Approach



ADL Phase 1

- Build ADL as Category 3 software
  - Full Cat 3 development, including requirements definition, design, coding, and test
- Assess likely savings to Science-to-Operation code conversion for MIS algorithms through vendor use of ADL and associated practices
  - Provide ADL to NGAS to use during development of Deep Blue aerosol algorithm
  - NGAS delivered Deep Blue to Raytheon for sci-2-ops conversion
  - Perform all conversion efforts required to assess ADL savings
  - Write joint report with NGAS documenting results

ADL Phase 1 Results

- Steps performed to build ADA compliant Deep Blue “science code”
  - Hosting PGE04 at Space Park
  - Converting Deep Blue LUTs and global surface reflectance database MODIS binary files into IDPS formatted binaries
  - Using the PRO XML editor GUI supplied with the ADL to generate XML product description files for the Deep Blue LUTs and global surface reflectance databases
  - Manually create the Deep Blue algorithm configuration and guide list XML files
  - Manually create Imakefile to enable auto generation of algorithm I/O software
  - Auto generate Deep Blue ADL source code
  - Modify Deep Blue driver to strip out all MODIS specific I/O
  - Modify Deep Blue driver to map the ADL input and output pointers to the internal variables
  - Modify internal logic in Deep Blue algorithm to use VCM flags for pixel screening
  - Consolidate variable initialization of hard coded parameters into Deep Blue driver subroutine
  - Rewrite EDR aggregation to conform to VIIRS Aerosol EDR HCS
  - Run test cases
  - Compare output of test cases to MODIS Deep Blue product output

ADL Phase 1 Results

- Steps performed to convert Deep Blue “science code” to operational code
  - Initial assessment and port
  - I-P-O Design
  - I-P-O code and unit test
- Sci-2-Ops tasks not performed (unnecessary for assessment of ADL savings)
  - Error Handling/Data Quality Design
  - Error Handling/Data Quality Code and Unit Test
  - Optimization/Graceful Degradation Design
  - Optimization/Graceful Degradation Code and Unit Test

ADL Phase 1 Results

- Output comparisons (one of four granules)
  - Differences within IPAC

NPP001212077949 DEEP BLUE EDR RESULTS SUMMARY

Field Name	Difference Count	Total Pixel Count
AerosolOpticalDepth_at_412nm	1	19149
AerosolOpticalDepth_at_488nm	1	19142
AerosolOpticalDepth_at_672nm	95	3239
AerosolOpticalDepth_at_550nm	5	19172
SingleScatteringAlbedo_at_412nm	20	14137
SingleScatteringAlbedo_at_488nm	3	14070
SingleScatteringAlbedo_at_672nm	0	14274
AngstromExponent	1	12254
qf1_AOT_M1_Product_Quality	0	38400
qf1_AOT_M3_Product_Quality	0	38400
qf1_AOT_M5_Product_Quality	114	38400
qf1_AOT_550_Product_Quality	0	38400
qf2_SSA_M1_Product_Quality	0	38400
qf2_SSA_M3_Product_Quality	0	38400
qf2_SSA_M5_Product_Quality	0	38400
qf2_Angstrom_Exponent_Product_Quality	0	38400
AOT_Scale	0	1
AOT_Offset	0	1
SSA_Scale	0	1
SSA_Offset	0	1
ANGEXP_Scale	0	1
ANGEXP_Offset	0	1

NPP001212077949 DEEP BLUE IP RESULTS SUMMARY

Field Name	Difference Count	Total Pixel Count
AerosolOpticalDepth	0	703504
AerosolOpticalDepth	0	703430
AerosolOpticalDepth	0	29420
AerosolOpticalDepth	58	704196
SingleScatteringAlbedo	1317	336245
SingleScatteringAlbedo	1317	336245
SingleScatteringAlbedo	0	336245
AngstromExponent	0	704119

ADL Phase 1 Results

- Deep Blue algorithm sci-2-ops conversion effort (without ADL) was estimated using standard CERs
  - Total effort 2,338 hrs
- Total effort was allocated to the five sci-2-ops steps
- Actual sci-2-ops hours (with ADL) were recorded for each of the steps, with extrapolation for sub-steps that were not performed
- Hours were compared and total efficiency calculated

TOTAL EFFICIENCIES

	Expected Hrs	Extrapolated/ Estimated Hrs	Efficiency
Port	164	16	90%
IPO Design	210	28	87%
IPO CUT	772	79	90%
DQ/EH Opt/GD Design	327	306	6%
DQ/EH Opt/GD CUT	865	699	19%
Total:	2,338	1,129	**52%**

ADL Phase 2

- The original goal of ADL was to provide algorithm developers with a tool that reduces the cost of the science-to-operational conversion effort
- Phase 1 accomplished this, with a multi-platform, endian neutral tool
  - Supports AIX, linux, and Windows platforms
  - Big endian or little endian
- Phase 2 expands ADL to become useful for two-way algorithm conversion efforts
  - Allow users to execute, test, and modify existing operational code on their home systems
  - Allows rapid incorporation of updated algorithms back into operational system

ADL Phase 2

- Tasks
  - Support spacecraft diary RDRs (Phase 1 used TLEs)
  - Convert ancillary files into appropriate IDPS gridded and granulated formats
  - Update operation code to ADL compatibility
    - Funding limitations prioritized to ATMS, VIIRS, and OMPS SDRs and VCM
    - Generalized endian-independent application packet reader
- Schedule
  - May 1, 2010 through Sept 30, 2010
- Other enhancements desired (but not currently funded)
  - Read from HDF-5 formatted files
  - Metadata handling
  - Chaining ability
  - Little-endian CrIS SDR

ADL Science Algorithm Developer Strategy

- Algorithm Development steps with ADL:
  1. Define the format of all algorithm inputs, SDR, IP, AUX (LUT), GranAnc, Geolocation using the ADL XML GUI tool
     - **Tool produces XML that will meet schema for operational system**
  2. Define the format of all algorithm outputs using ADL XML GUI tool
  3. After inputs and outputs are defined via XML an algorithm configuration guide needs to be created that specifies the input and output products for the algorithm via group names specified when creating the XML products
     - **The algorithm configuration guide will contain the algorithm name and inputs/outputs specified by group names that map to a specific file name**
  4. Compile the ADL software, which will auto generate the dictionary source code from the XML product definitions and create corresponding C++/Fortran structures/modules
     - **The compile will also automatically produce an auto generated derived algorithm class that will handle all of the I/O and pointer assignment automatically**

ADL Science Algorithm Developer Strategy

- Algorithm Development steps with ADL (continued):
  5. A derived algorithm class will also be created if one does not exist as a starting point for inserting the science code. Subsequent recompiles will not do this step, once a file exists
  6. Algorithm developer proceeds to develop the science algorithm and makes use of the available derived algorithm pointer data members
  7. If during development new inputs are added or taken away in the algorithm configuration file a recompile will be necessary to auto generate the algorithm class
     - **If inputs are taken away, they will be removed from the auto generated algorithm class which may subsequently cause compile failures in the derived algorithm class if references still exist**

ADL Operational Developer Strategy

- Operational Development steps with ADL:
  1. **Operational developer is delivered software for algorithm that was developed on ADL platform**
     - **Operational developer compiles and runs algorithm software in local ADL environment and confirms software produces results within specification (port)**
  2. **Operational developer inserts new XML for input and output definitions into operational system if products don't currently exist under appropriate sensor directory**
  3. **Operational developer copies the dropped algorithm configuration guide and updates the file names specified to be actual operational collection short names**
  4. **Operational developer inserts binary data input files provided with drop into DMS with proper metadata for operational system, granule id, granule version, effectivity, etc.**
     - **Operational developer may need to convert dropped input data if not in operational format, or format has changed**
       - **Science algorithm developer should try to develop algorithm with correctly formatted operational data, otherwise the cost of doing Sci2Ops is increased to accommodate the data conversion needed**

ADL Operational Developer Strategy

- Operational Development steps with ADL (continued):
  5. **Operational developer places all algorithm software except for auto generated files into appropriate algorithm folder**
  6. **Operational developer updates Makefile to point to operational auto generation software for the algorithm**
  7. **Operational developer compiles dictionary with new XML input and output items and generates operational C++/Fortran structures and produces auto generated derived algorithm class**
  8. **Operational developer runs algorithm and validates results are within specification**
  9. **Operational developer adds operational functionality to the algorithm by updating the "empty" implementation algorithm delegate class that was created for metadata, data quality notification, graceful degradation, etc. but isolating these operational changes in the class**
     - **Operational developer should only make science code changes to the derived algorithm class, this allows for the plug and play of the algorithm between ADL and the operational environment**
       - **It is conceivable that Operational developers could enhance the science or output format by implementing quality flags as an example that weren't in the original algorithm drop**

Return ADL Operations Code to Developers

- Process of returning updated code from operations to algorithm developers:
  1. **Operational developer makes updates to algorithm processing code to implement quality flags and/or bug fixes. Code changes that are for operational aspects must be made to the delegate class to segregate the ops changes to allow for plug and play**
  2. **Operational developer re-tests the algorithm in ADL environment by taking the updated derived algorithm and any XML changes to input or output formats**
  3. **Operational developer compiles ADL dictionary with updated XML input and output items and generates C++/Fortran structures and produces auto generated derived algorithm class**
  4. **Operational developer runs algorithm in ADL environment and validates results are within specification of the operational results**
  5. **Operational developer prepares updated ADL software delivery to algorithm developer**
     - **Algorithm developer will need to merge code changes with any new changes (if working on parallel changes from operations) and potentially resolve any XML input/output format issues**
  6. **Algorithm developer runs delivered code in local ADL environment (re-port)**

ADL Benefit Summary

- Use of ADL will reduce algorithm development costs by
  - Providing standardized toolkits and development GUIs for interfaces
  - Providing standardized framework for “I to P” and “P to O” interfaces
- Use of ADL will reduce operational implementation costs by
  - Ensuring “plug and play” compatibility with operational baseline
  - Reduce time and effort required to make an algorithm operational
- ADL framework was created for use by algorithm developers to facilitate future business through:
  - Plug and play compatibility between the algorithm developers and operations
  - Less cost and effort to operationalize science algorithms

Use of ADL will benefit any organization creating, modifying, testing, or integrating algorithms into an operational system