N95-16457

A Software Tool for Dataflow Graph Scheduling

Robert L. Jones III
NASA Langley Research Center

A graph-theoretic design process and software tool is presented for selecting a multiprocessing scheduling solution for a class of computational problems. The problems of interest are those that can be described using a dataflow graph and are intended to be executed repetitively on multiple processors. The dataflow paradigm is very useful in exposing the parallelism inherent in algorithms. It provides a graphical and mathematical model which describes a partial ordering of algorithm tasks based on data precedences. That is, some tasks must execute in a particular order whereas other tasks may execute independent of other tasks. Dataflow graph nodes represent schedulable tasks and edges represent the data dependencies between the tasks. Analytical analysis of the dataflow graph is possible for many digital signal processing (DSP) and control law algorithms which are deterministic. For determinism, the model is applicable to a class of dataflow graphs where the time to execute tasks are assumed constant from iteration to iteration when executed on a set of identical processors. Also, it is assumed that the dataflow graph is data independent. Any decisions present within the computational problem must be contained within the graph nodes rather than described at the graph level. Special transitions called sources and sinks are also provided to model the input and output data streams of the task system. The presence of data is indicated by marking the dataflow graph with tokens. The graph transitions through markings as a result of a sequence of node firings. A node is enabled for firing when a token is available on every input edge of the node, indicating that the task has all of its operands. When the node fires, it encumbers one token from each of its input edges, delays an amount of time equal to the task latency, and then deposits one token on each of its output edges. Sources and sinks have special firing rules in that sources are unconditionally enabled for firing and sinks consume tokens, but do not produce any. By analyzing the dataflow graph in terms of its critical path, critical circuit, dataflow schedule, and the token bounds within the graph, the performance characteristics and resource requirements can be determined *a priori*.

As for any mathematical model, there is a need for efficient software tools which facilitate the use of the model in solving problems. A software program, referred to as the Dataflow Design Tool, was developed at Langley to apply the dataflow model and design multiprocessor solutions for spaceborne applications. The tool was written in C++ for Microsoft Windows 3.1 or NT can be hosted on an i386/486 personal computer or compatible. The Design Tool takes input from a text file which specifies the topology and attributes of the dataflow graph. A Graph Tool was developed to facilitate the creation of the graph text file. The various displays and features are shown to provide an automated and user-interactive design process which facilitates the selection of a multiprocessor solution based on dataflow analysis. Performance metrics determined automatically by the Dataflow Design Tool include the minimum time to execute all tasks for a given computation (schedule length), the minimum time between graph input and the corresponding output (TBIOlb), the minimum graph-imposed iteration period (To), and the minimum time between outputs (TBOlb). Also, the tool determines if tasks can be delayed a finite amount of time without degrading performance, referred to as slack time. Since the edges imply the physical storage of data, the tool can calculate the minimum data buffers required for proper

sharing of data between tasks. In addition to numerical performance metrics, the tool graphically portrays system behavior using Gantt charts and resource envelopes. The Single Graph Play displays the steady-state task schedule associated with a single computation, and the Total Graph Play displays the periodic, steady-state task schedule over a single iteration period.

The analysis and multiprocessor mapping of a finite impulse response (FIR) filter is illustrated. A linear phase FIR filter is selected since it requires half the number of multiplies of other FIR realizations. DSP problems are very suitable for dataflow analysis since they are typically described as signal flow graph. One can easily translate signal flow graphs to dataflow graphs by locating the computations (addition and multiplication) and representing unit delays (inverse z terms) with initial tokens. Once the filter has been captured into the Graph Tool it can be analyzed by the Dataflow Design Tool to expose the inherent parallelism and determine graph-theoretic performance bounds. Since there are many realizations of the same filter, characterized by different dataflow graphs, the Dataflow Design Tool can be useful in determining which realization exposes the most parallelism. The SGP shows that some of the additions can execute in parallel (C1 through C4), enabling the parallel execution of the multiplies, and finally the sequential summation to generate the output sample. The SGP bars are shaded to depict the read, process, and write activities of the processor, and the hollow bars denote slack time associated with some tasks. In addition to the parallel concurrency, the TGP shows pipeline concurrency that may be exploited. In this example, the TGP shows that at most 4 different data samples may be computed within a sampling period of 224 time units. The Total Resource Envelope shows that 10 processors are required to achieve this level of throughput. The dataflow analysis applied to the dataflow graph and portrayed in the graph play diagrams assume infinite resources (processors and memory) so that the exposed parallelism is limited only by the data precedences. If there is not enough resources to exploit the inherent parallelism, the schedule must be optimized. As an example, lets assume that a fully-static schedule (i.e., a task will execute on the same processor for every iteration) on 8 processors is desirable to minimize run-time overhead. The Dataflow Design Tool shows that such a solution can be achieved by inserting two additional "artificial" data dependencies and increasing the sampling period to 260 time units. The tool can also display the periodic memory accesses within a periodic schedule. Such an assessment may be useful to optimize the schedule based on the limited bandwidth between processors or processors and memory. Once a desirable solution is obtained, the tool can summarize the scheduling constraints in terms of earliest start (ES), latest finish (LF), and slack time. The summary of run-time requirements include task instantiations (INST) defined as the number of processors a task will have to execute on simultaneously for different data sets. For a fully-static schedule, one would expect all instantiations to be 1 as shown. Also, the buffer sizes (QUEUE) for shared data is given along with the number of initially empty buffers (OE) and the number of initially full buffers (OF) due to initial data.

In summary, the dataflow paradigm provides a general model suitable in exposing parallelism inherent in algorithms as fine-grain as filters to more computationally complex algorithms where a node might represent an entire filter. When the algorithm is deterministic, the Dataflow Design Tool can analytically determine the steady-state behavior, performance bounds, scheduling constraints, and resource requirements. By permitting the user to insert artificial data dependencies, the dataflow schedule can be optimized to match resource requirements with resource availability.

A Software Tool for Dataflow Graph Scheduling

June 15, 1994

Robert L. Jones III

NASA Langley Research Center
Hampton, Virginia

Outline

- Functional Overview
- Analysis of a DSP Filter
- Static Scheduling and Optimization
- Summary

Dataflow Design Tool

Task System, ($\mathcal{T}$, $\mathcal{L}$, $\prec$, $\mathcal{M}_0$)

- $\mathcal{T}$ Set of Tasks
- $\mathcal{L}$ Fixed Task Latencies
- $\prec$ Partial-Order on $\mathcal{T}$
- $\mathcal{M}_0$ Initial State

Dataflow Graph (DFG)

Performance Bounds

- Schedule Length, ω
- Time Between Input & Output, $TBIO_b$
- Minimum Iteration Period, T_0
- Time Between Outputs, TBO_b
- Slack
- Processor Utilization

DFG → Graph Analysis

Run-Time Requirements

- Task Instantiations
- Processor Requirement
- Data Buffers
- Artificial $\prec$, Control Edges

Dataflow Graph

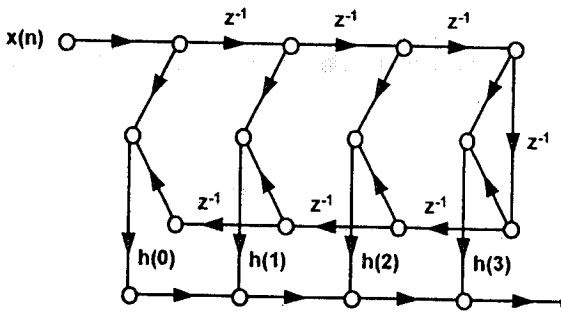
- Nodes Represent $\mathcal{T}$
- Edges Describe $\prec$
- Tokens Indicate Presence of Data
- Initial Marking = $\mathcal{M}_0$

Graphical Displays

- Gantt-Chart Task Execution
  - Single Iteration (SGP)
  - Periodic Execution (TGP)
- Resource Envelopes

Eight-Order, Linear Phase FIR Filter

Direct Form Signal Flow Graph

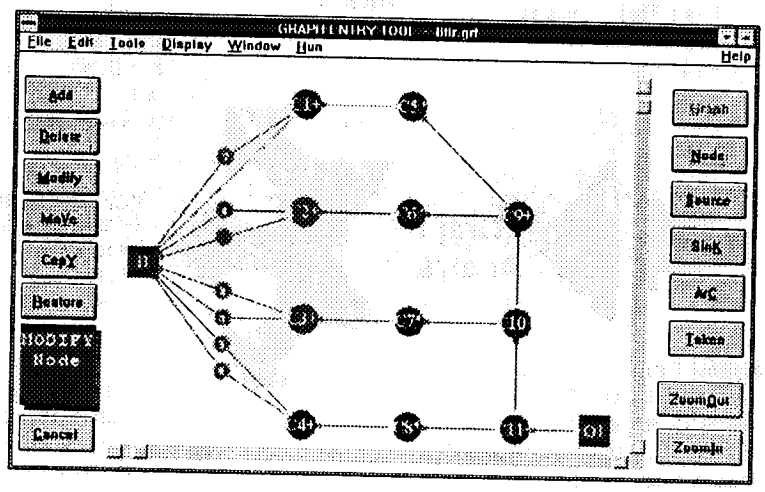


Tasks	Instructions
C1+	$x_0 = x(n) + x(n-7)$
C2+	$x_1 = x(n-1) + x(n-6)$
C3+	$x_2 = x(n-2) + x(n-5)$
C4+	$x_3 = x(n-3) + x(n-4)$
C5*	$x_4 = x_0 * h(0)$
C6*	$x_5 = x_1 * h(1)$
C7*	$x_6 = x_2 * h(2)$
C8*	$x_7 = x_3 * h(3)$
C9+	$x_8 = x_4 + x_5$
C10+	$x_9 = x_6 + x_8$
C11+	$y(n) = x_7 + x_9$

A DSP signal flow graph is a *Dataflow Graph* where the z^{-1} unit delays can be modeled with initial tokens. Thus, run-time implementation of *delay* does not incur any overhead. Unit delays are simply implemented by initializing FIFO queues used for intermediate data.

Dataflow Graph Capture of FIR Filter

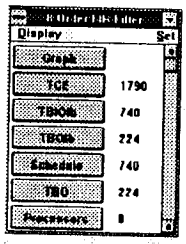
Graph Tool



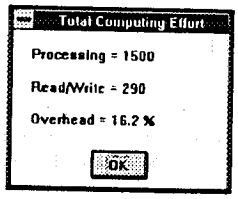
Multiprocessor Implementation Example

Assumptions: Shared memory with no contention
Multiplies take 200 *time units*
Additions take 100 *time units*
One-operand read/writes take 10 *time units*
Two-operand read/writes take 20 *time units*

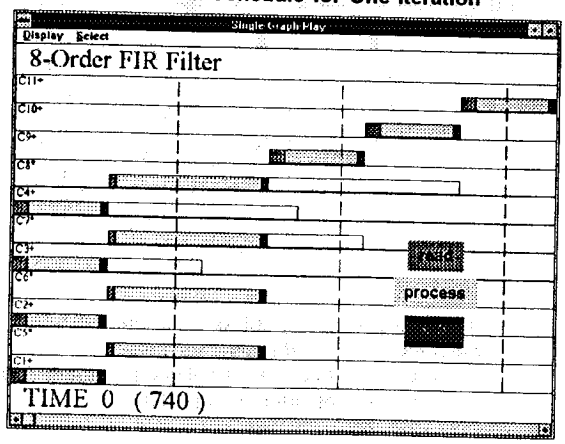
Performance



Division of Effort

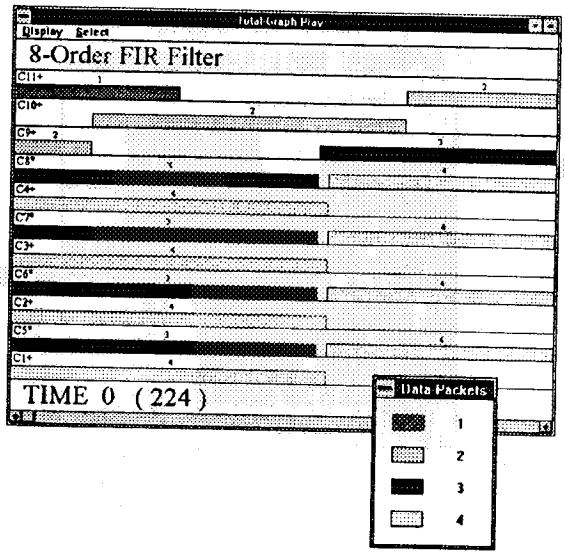


Data-Driven Schedule for One Iteration

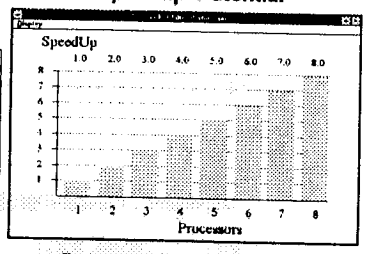


Exposing the Parallelism in the FIR Filter

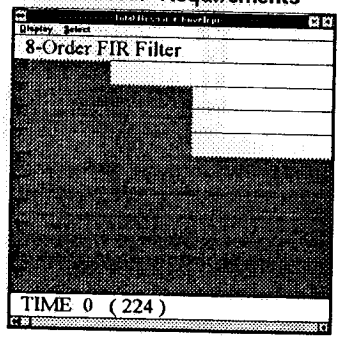
Steady-State Periodic Schedule



Speedup Potential



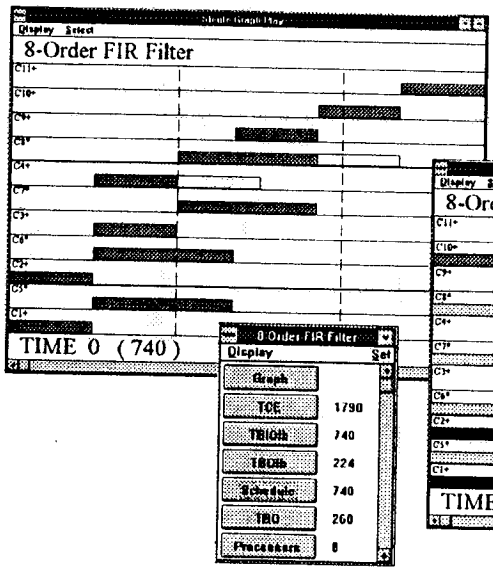
Processor Requirements



Optimization for 8 Processors

A fully-static schedule is desired for minimum run-time overhead.

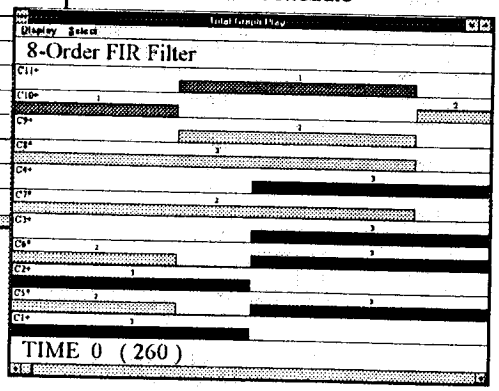
Single-Iteration Schedule



Artificial Data Precedences

C1+ ≺ C4+
C2+ ≺ C3+

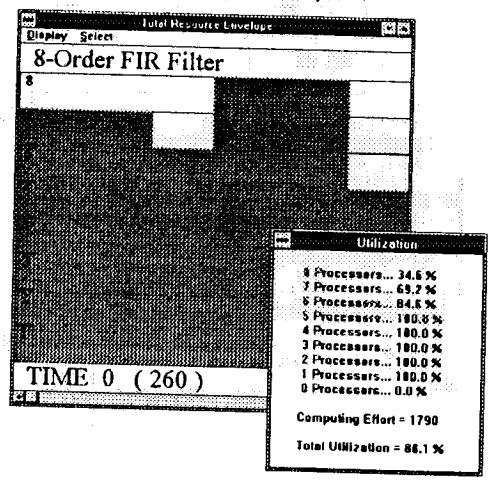
Periodic Schedule



Fully-Static Processor Requirements

Sampling Period = 260 *time units*

Total of 8 DSP Chips are Required



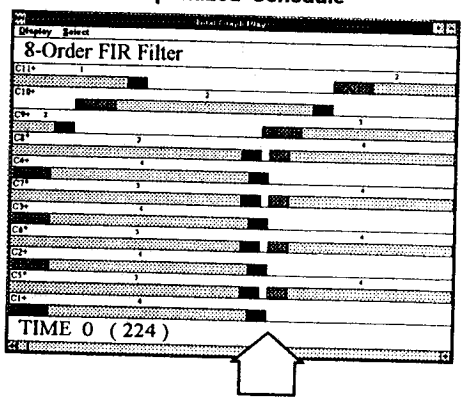
Processor Assignments

P1	{C1+, C4+}
P2	{C2+, C3+}
P3	{C5*}
P4	{C6*}
P5	{C7*}
P6	{C8*}
P7	{C9+, C10+}
P8	{C11+}

Analysis of Memory Access

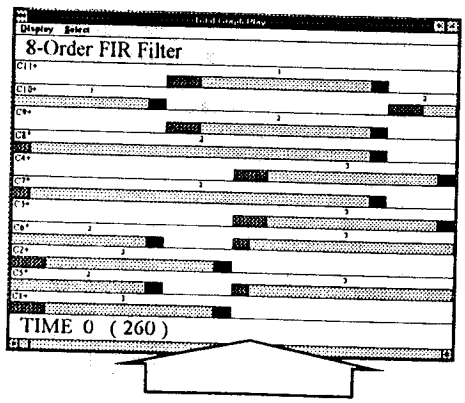
Optimized schedule has better distribution of memory accesses which e.g., can be accomodated with 6 independent communication ports in the TMS320C40's.

Unoptimized Schedule



Too many localized memory references!

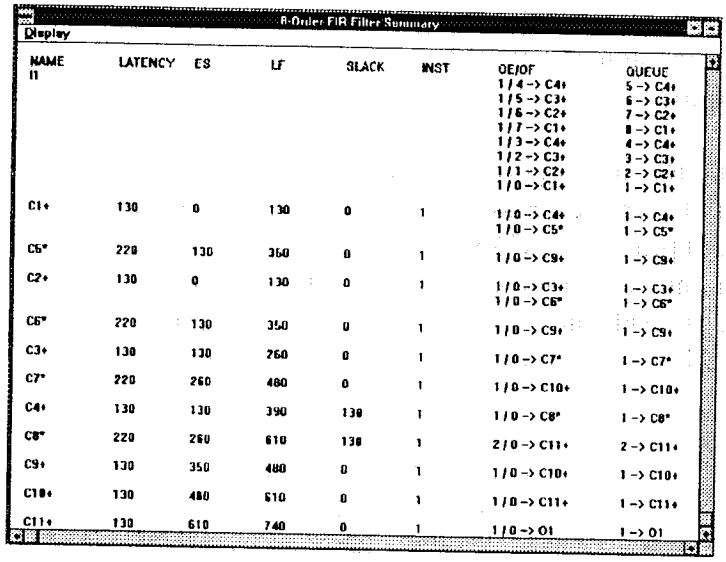
Optimized Schedule



Memory references are more evenly distributed.

Summary of Fully-Static Multiprocessor Solution

FIR Filter



8-Order FIR Filter Summary

Display

NAME II	LATENCY	ES	LF	SLACK	INST	OE/OF	QUEUE
						1 / 4 -> C4+ 1 / 5 -> C3+ 1 / 6 -> C2+ 1 / 7 -> C1+ 1 / 3 -> C4* 1 / 2 -> C3* 1 / 1 -> C2* 1 / 0 -> C1*	5 -> C4+ 6 -> C3+ 7 -> C2+ 8 -> C1+ 4 -> C4* 3 -> C3* 2 -> C2* 1 -> C1*
C1+	130	0	130	0	1	1 / 0 -> C4* 1 / 0 -> C5*	1 -> C4* 1 -> C5*
C5*	220	130	360	0	1	1 / 0 -> C9+	1 -> C9+
C2+	130	0	130	0	1	1 / 0 -> C3* 1 / 0 -> C6*	1 -> C3* 1 -> C6*
C6*	220	130	350	0	1	1 / 0 -> C9+	1 -> C9+
C3+	130	130	260	0	1	1 / 0 -> C7*	1 -> C7*
C7*	220	260	480	0	1	1 / 0 -> C10+	1 -> C10+
C4+	130	130	390	130	1	1 / 0 -> C8*	1 -> C8*
C8*	220	260	610	130	1	2 / 0 -> C11+	2 -> C11+
C9+	130	350	480	0	1	1 / 0 -> C10+	1 -> C10+
C10+	130	480	610	0	1	1 / 0 -> C11+	1 -> C11+
C11+	130	610	740	0	1	1 / 0 -> O1	1 -> O1

Summary

- Dataflow provides a general model of computation capable of exposing fine- and large-grain parallelism.
- Design Tool provides analytic, compile-time prediction of:
  - Steady-state behavior
  - Graph-theoretic performance bounds
  - Iterative run-time scheduling criteria
- Permits inclusion of artificial precedences for optimization.
- Facilitates selection of static run-time schedules.

Use of Software through Pictures on CERES

The CERES team has been using the Yourdon/DeMarco Structured Analysis/Structured Design methodology to develop the data management system for producing higher order science data products from CERES instrument data. As part of this effort, the team is using the Software through Pictures CASE tool to automate portions of the methodology. This presentation addresses the team's experiences with the selected methodology and CASE tool, describes lessons learned, and provides recommendations for other teams contemplating the use of structured methodologies and CASE tools.

Software Engineering methodologies can help developers create systems in less time with higher reliability and quality by providing tools for managing the complexity inherent in software systems and development programs. CASE tools can facilitate using a methodology by providing tools for creating and maintaining requirements and design models, automating consistency and completeness checking, and automating much of the bookkeeping associated with following the methodology. This allows developers to focus on the creative aspects of software design and development.

Overall, our experience on CERES has been that structured methodologies and CASE tools prove useful in creating, maintaining, and documenting high quality requirements and analysis products. Although the learning curves associated with these tools require an investment in time and training early on, the benefits to be gained are well worth the effort and our productivity continues to increase as we become more familiar with the methodology.

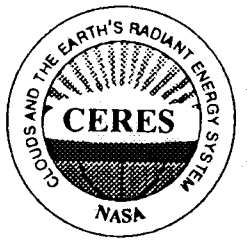
To date, the CERES data management team has used the tool to model more than 130 data products down to the level of atomic variables, define each data element in terms of type, units, accuracy, and number of bits, and create documentation from the information stored in the models. Since the CERES system is primarily a science data processing system which generates more than 5 terabytes of data per month, focusing on the system's data products has led to a deeper understanding of processing needs and resulted in higher quality functional requirements. Furthermore, the graphical editors and consistency checking features provided by the tool have allowed the team to rapidly iterate through the modelling process in less time than would have been required without the tool.

The data management team is currently analyzing system functional requirements by modelling the functionality needed to process instrument and higher order science data. Here again, the tool speeds up the process of iterating on the model to converge on a final solution. In addition, the tool has allowed the team to automatically produce software requirements documents in a standard format from information contained in the CASE tool database.

We have incorporated several customizations in order to tailor the CASE tool to support the specific processes employed on CERES. These customizations include creating templates for producing CERES-specific documentation, enhancing the CASE tool main menu, and integrating the CASE tool with the FrameMaker desktop publishing package. The CASE tool is supplied with templates for producing documentation that complies with military software standards. Since these standards were not appropriate for NASA publications, we developed templates for

several documents including a Software Requirements Document, Data Products Catalog, and Data Dictionary as well as several utilities to provide hard copies of details stored in the tool's database for developer's use in reviewing their models. We have also modified the tool's main menu to simplify the user interface for creating documents. Finally, there are several places in the tool where the developer adds detail to the requirements or design model by entering free form text. These items include functional descriptions, data product descriptions, and interface descriptions. The CASE tool only supports ASCII text and, since much of our processing is described in terms of equations, tables, and graphics this restriction limited our ability to fully describe the necessary processing. Therefore, we have modified the tool to allow the use of FrameMaker (desktop publishing/word processor) for entering descriptions of functions, data products, and interfaces. This allows a designer to include any combination of text, graphics, tables, and equations in these descriptions which are then included directly into the documentation produced using the tool.

Our experience indicates that when combined with well-structured methodologies, CASE tools can provide a important component of a development environment which helps designers create software products with higher quality in less time. However, the key to achieving productivity gains is the process used to design the software. The processes incorporated in structured analysis and structured design provide a sound framework for creating complex software systems and must be adopted in order to derive any benefits from the use of automated tools such as Software through Pictures.

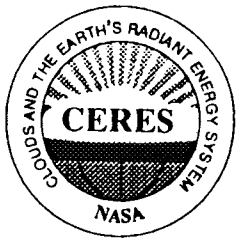


Use of Software through Pictures on CERES

The Role of Computers in LaRC R&D Workshop
June 15-16, 1994

Troy Anselmo
Science Applications International Corporation

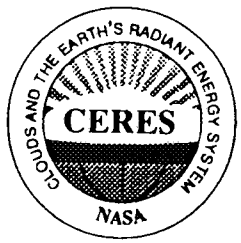
INTRODUCTION



- CERES Overview
- Software Development Methodology
- CASE Tool Capabilities and Configuration
- Experiences to Date
- Lessons Learned/Recommendations

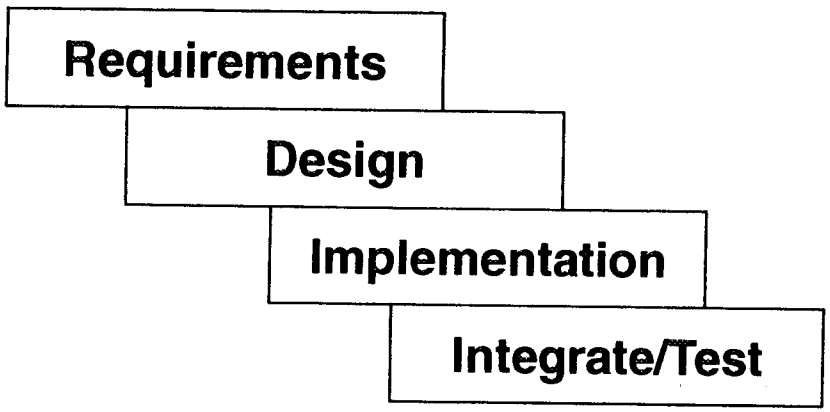
SAIC
An Employee-Owned Company®

CERES OVERVIEW



- Scientific Data Processing Application
- Approximately 500K Source Lines of Code
- Organized into 12 Subsystems (CSCIs)
- Generates More Than 5 TeraBytes of Data per Month
- Operates within the EOSDIS Environment
- Languages include FORTRAN, C, Ada

METHODOLOGY

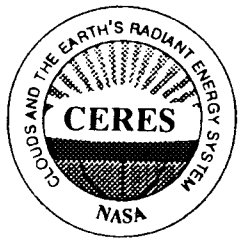


Build the Right Product

Build the Product Right

- Methodology Consists of:
  - Notation - Capture Models, Reason about Models, Communicate to Others
  - Process - Procedures for Creating Models

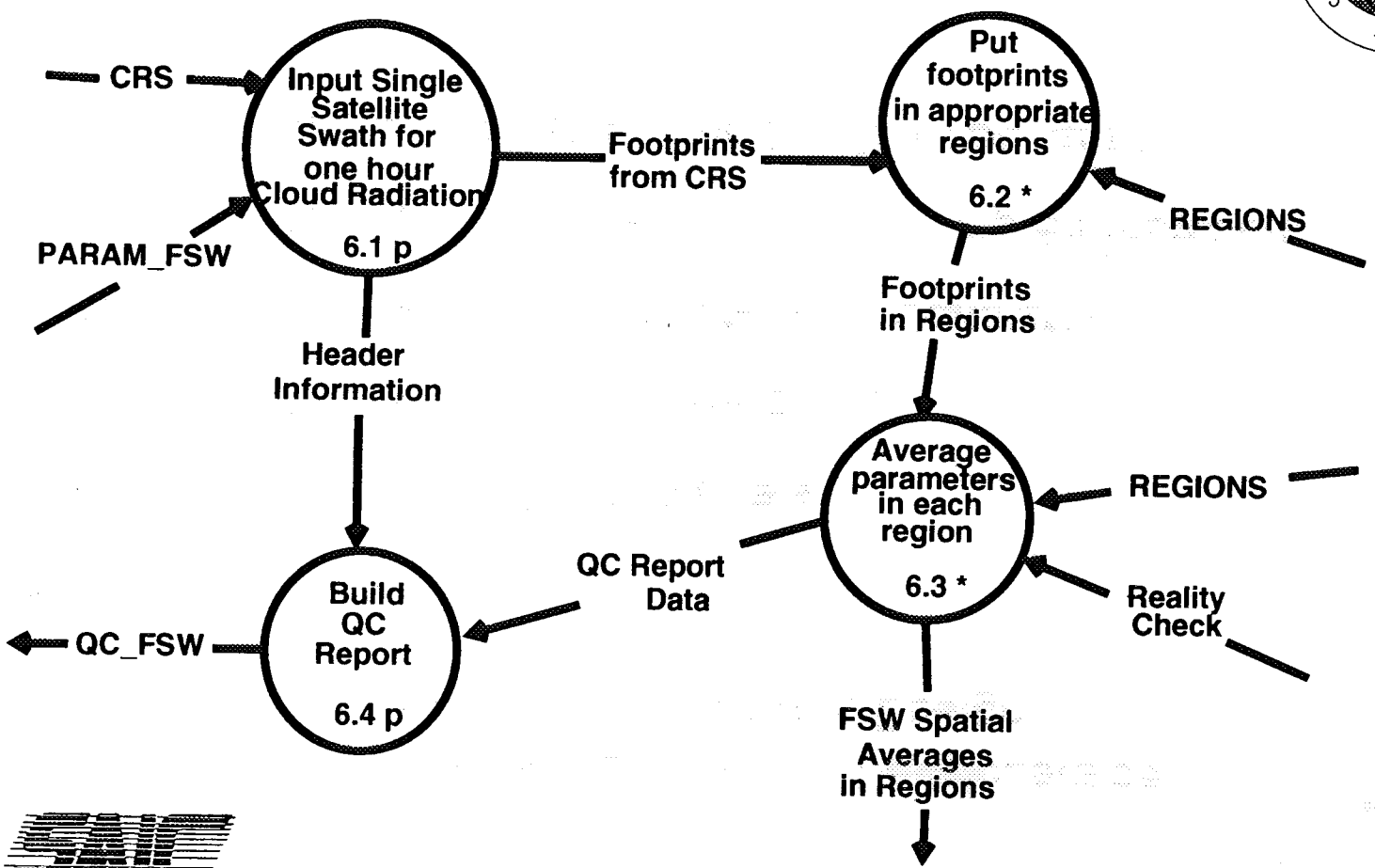
METHODOLOGY (cont'd)



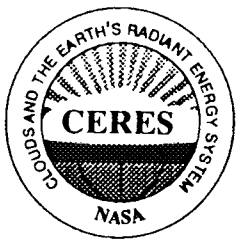
- Structured Analysis/Structured Design
  - Model Based Approach
  - Emphasis on Early Life Cycle Phases
  - Requirements - Model functionality
  - Design - Models Architecture of Solution

SAIC
An Employee-Owned Company®

METHODOLOGY (cont'd)

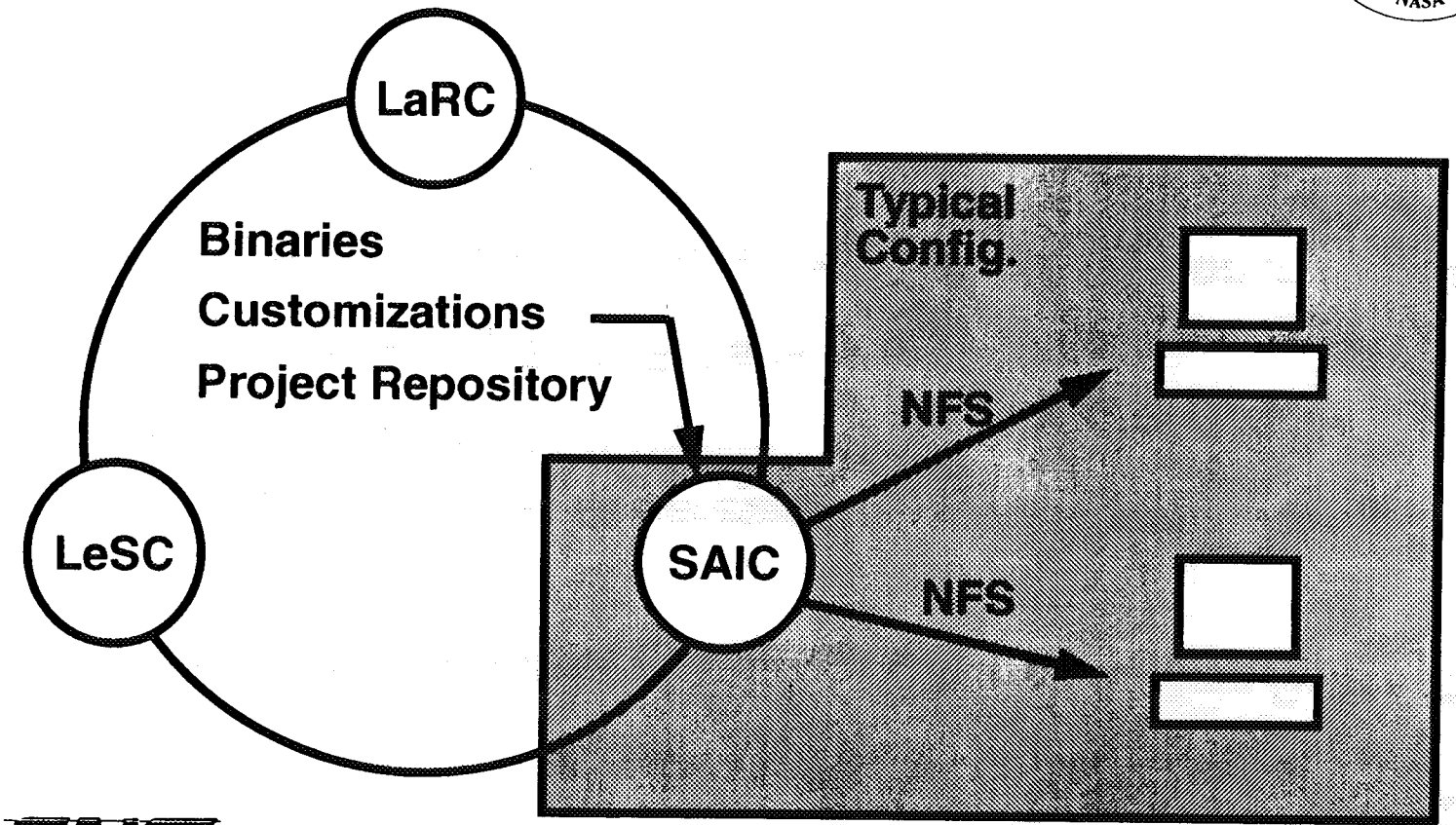


CASE TOOL CAPABILITIES

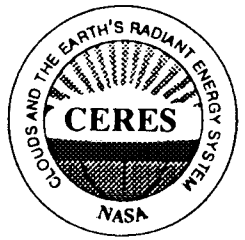


- Automate Portions of Methodology so Developers can Focus on Creative Aspects of Software Design
- Provide Tools to:
  - Rapidly Create and Modify Models
  - Capture Models in Central Repository
  - Check Model Validity (Completeness, Consistency)
  - Support Multiple Developers in Work Group Environment
  - Create Documentation from Models in Repository

CASE TOOL CONFIGURATION

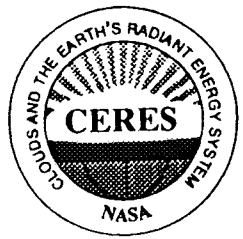


EXPERIENCES TO DATE



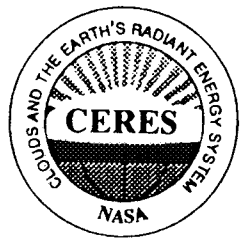
- **Achievements**
  - **Data Products Modelled (incl. Data Structure and Data Description Details)**
  - **Data Product Catalogs Generated from Data Models (Sizing Analysis Computed by Tool)**
  - **Currently Modelling Each Subsystem**
  - **Automatically Produce Requirements Documentation in Standard Format for Each Subsystem**

EXPERIENCES (cont'd)



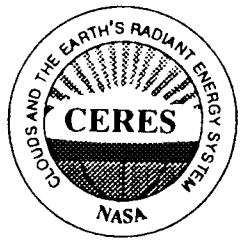
- Customizations
  - CERES Specific Document Templates
  - Main Menu Changes to Facilitate Document Generation
  - Integration with FrameMaker to support Graphics, Tables, and Equations
- Positive Results
  - + Validation of Interfaces Between Subsystems
  - + Communication of Functional and Data Models Among Team Members (incl. Science Team)
  - + Allows Rapid Iterations on Diagrams to Converge on Solutions

EXPERIENCES (cont'd)



- Issues
  - Multiple-Site Configuration Complicated System Administration Functions
  - Document Definition in Parallel with Template Development Resulted in Excessive Template Iterations
  - "Loose" Configuration Management of Customizations Created Synchronization Problems Among Multiple Sites
  - Lack of CASE/Methodology Expertise at Each Site Slowed CASE Tool Adoption

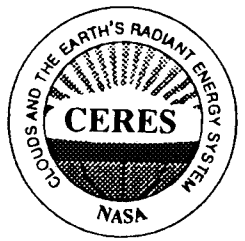
EXPERIENCES (cont'd)



- Quality Action Team Established by NASA to Address Concerns, Improve CASE Tool Use
- Membership From All Organizations
- Results to Date Have Been Very Positive
  - Enhanced Understanding and Awareness of Concerns Among Organizations
  - Simplified System Administration Process and Configuration
  - Establish Forums for Information Dissemination, Identified Training Needs, Conducted Training
  - Improved Development, Test, CM Process for Customizations

SAIC
An Employee-Owned Company®

LESSONS LEARNED/RECOMMENDATIONS



- CASE Tool Introduction Represents Potential Culture Change
  - Strong Management Support Required
  - Steering Committee Useful for Coordinating Adoption Process
  - Timely Dissemination of Information Necessary, Exploit Electronic Communications Media (e-mail, bulletin boards, WWW)
- Methodology is *Key* Element, Training is Critical
- CASE is Engineering Tool, Documentation is By-Product
- Customizations Represent Development of Utility Codes, Should Use Structured Development Process

SAIC
An Employee-Owned Company®

SESSION 4 Solutions of Equations

Chaired by

Olaf Storaasli