AFRL-RI-RS-TR-2013-078

DATAFLOW-BASED IMPLEMENTATION OF LAYERED SENSING APPLICATIONS ON HIGH-PERFORMANCE EMBEDDED PROCESSORS

UNIVERSITY OF MARYLAND

MARCH 2013

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2013-078 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/ S /

STANLEY LIS
Work Unit Manager

/ S /

RICHARD MICHALAK
Acting Tech Advisor, Computing
& Communication Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Service, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
MARCH 2013	FINAL TECHNICAL REPORT	OCT 2010 – OCT 2012

4. TITLE AND SUBTITLE
DATAFLOW-BASED IMPLEMENTATION OF LAYERED SENSING APPLICATIONS ON HIGH-PERFORMANCE EMBEDDED PROCESSORS

5a. CONTRACT NUMBER FA8750-11-1-0049
5b. GRANT NUMBER N/A
5c. PROGRAM ELEMENT NUMBER 62788F

6. AUTHOR(S)
Chung-Ching Shen, Shenpei Wu, Lai-Huei Wang, Stephen Won, Kishan Sudusinghe, and Shuvra Bhattacharyya

5d. PROJECT NUMBER T2KA
5e. TASK NUMBER IL
5f. WORK UNIT NUMBER PP

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Maryland, College Park
7965 Baltimore Ave
College Park, MD 20742

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/RITB
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
N/A

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TR-2013-078

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Layered Sensing is an emerging paradigm for critical defense technologies in which heterogeneous methods for sensing, communication, signal processing, and information exploitation must be integrated with high flexibility, reliability and efficiency. New design methodologies and software tools will be required to handle the complexity of layered sensing applications, and allow designers to explore trade-offs among alternative sensing and exploitation strategies while satisfying their stringent performance and power consumption constraints, and exploiting the capabilities of state-of-the art embedded processing platforms. In this project, we have developed new dataflow-based technology and associated design tools for high-productivity, high-confidence design and optimization of layered sensing software. These tools are geared towards systematically exploring and optimizing interactions across application behavior, operational context, high performance embedded processing architectures, and implementation constraints.

15. SUBJECT TERMS
Layered sensing, signal processing, embedded systems, dataflow.

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
					STANLEY LIS
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPONE NUMBER (*Include area code*)
U	U	U	UU	33	**N/A**

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

TABLE OF CONTENTS

LIST OF FIGURES II
LIST OF TABLES III
1. SUMMARY 1
2. INTRODUCTION 2
3. METHODS, ASSUMPTIONS, AND PROCEDURES 3
3.1. PARAMETERIZED SCHEDULING USING TOPOLOGICAL PATTERNS 3
3.2. SCALABLE SCHEDULE TREES 3
3.3. IMPROVED GPU-TARGETED SYNTHESIS TOOL 4
3.4. LIBRARY COMPONENTS AND APPLICATION EXAMPLE 6
3.4.1. Application: Image Registration 6
3.5. EMULAB SOFTWARE TOOL 8
3.5.1. NT-SIM CASE STUDY: VISUAL SENSOR NETWORK 10
3.6. IMPROVEMENTS TO DIFML 13
4. RESULTS AND DISCUSSION 15
4.1. CASE STUDY: IMAGE REGISTRATION 15
4.1.1. EVALUATION FOR PERFORMANCE ACCELERATION 15
4.1.2. EVALUATION IN TERMS OF CODING EFFICIENCY 16
4.2. CASE STUDY: VISUAL SENSOR NETWORK 17
5. CONCLUSION 17
6. PUBLICATIONS 17
7. REFERENCES 19
A. APPENDIX–PROJECT DELIVERABLES 21
A.1. INTRODUCTION TO DELIVERABLE ORGANIZATION 21
A.2. Instructions for Deliverable Installation and Startup 22
A.3. Instructions for the Demonstrations 23
A.3.1. Demo for Image Registration using TDIF 23
A.3.2. Demo for SIFT Visual Sensor Network using NT-SIM 24
A.3.3. Demo for Image Registration using Topological Patterns 24
A.3.4. Demo for cascade Gaussian filtering using SST plug-in 25
A.3.5. Demo for Image Registration using DIFML 25
LIST OF ACRONYMS 26

LIST OF FIGURES

Figure 1: TDIF design flow. 5
Figure 2: Design flow of the targeted image registration application. 6
Figure 3: Cascade Gaussian filtering. 7
Figure 4: Illustration of the interaction between dataflow applications and network simulations in NT-Sim. 9
Figure 5: A dataflow graph model of SIFT-based feature detection and image registration across a network. 10
Figure 6: The topology represented by the Tcl script for the SIFT sensor network. 13
Figure 7: LOC evaluation results. 15
Figure 8: (Clockwise from top left) Reference image, target image, and registered image from the simulated SIFT VSN. 16

LIST OF TABLES

Table 1: Performance comparison between CPU-targeted and GPU-targeted actors................................... 14
Table 2: Performance comparison for the overall application with and without GPU acceleration........ 14
Table 3: Experiments for comparison with GPU peak performance... 14

1. Summary

Below is a summary of accomplishments, listed by project task.

- Application Case Study: Image Registration: Accomplished. A rigorous case study was developed to demonstrate the impact of dataflow-based design techniques in the domain of image registration. In our case study, we quantitatively and qualitatively assessed a comprehensive dataflow-based methodology for developing, encapsulating, and integrating image registration functional components.
- Improved Software Synthesis for Optimized Implementation on State-of-the-Art GPUs (Graphic Processing Units): Accomplished. Based on the experimentation in our proposed image registration case study, we identified bottlenecks and other areas of improvement in our GPU-targeted software synthesis tools. We designed, implemented, evaluated and refined methods to address these limitations and further improve the efficiency of our dataflow-based software synthesis tools.
- GPU-targeted Dataflow Library for Image Registration: Accomplished. We developed a collection of dataflow library components for GPU-based image registration. This library allows designers to experiment with alternative image registration techniques (e.g., different registration metrics, preprocessing techniques, and optimization subsystems) in the context of an enclosing model-based design methodology.
- Formal Models for Representation and Transformation of Performance Optimization Configurations: Accomplished. To help support a wide range of performance optimization techniques, and to improve the interoperability of lower level code tuning techniques with system-level design methodologies, we developed dataflow-based intermediate representations for encapsulating structures for scheduling. We integrated these representations in our application case study to demonstrate their efficiency, and tune their application to the image registration domain.
- Improvements to DIFML (Dataflow Interchange Format Markup Language): Accomplished. We incorporated our new image registration component library into the DIFML software package and extended the test suite for the package to provide improved code coverage.
- Emulab Software Tool: Accomplished. We developed a new software tool that provides novel capabilities for experimenting with networked signal processing systems. Our tool integrates our dataflow-based design tool with Emulab-based ns-2 scripts and provides a flexible environment that allows designers to simulate systems comprehensively at both the node and network levels.

The organization of project deliverables is summarized in Appendix–Project Deliverable of this report along with instructions for usage and demonstration of the software deliverables.

Approved for Public Release; Distribution Unlimited.

2. Introduction

Layered Sensing is an emerging paradigm for critical defense technologies in which heterogeneous methods for sensing, communication, signal processing, and information exploitation must be integrated with high flexibility, reliability and efficiency. New design methodologies and software tools will be required to handle the complexity of layered sensing applications, and allow designers to explore trade-offs among alternative sensing and exploitation strategies while satisfying their stringent performance and power consumption constraints, and exploiting the capabilities of state-of-the art embedded processing platforms.

The objective of this research is to develop new dataflow-based technology and associated design tools for high-productivity, high-confidence design and optimization of layered sensing software. These tools are to be geared towards systematically exploring and optimizing interactions across application behavior, operational context, high performance embedded processing architectures, and implementation constraints.

In this project, we have developed an application case study of dataflow-based design in the domain of image registration. Image registration is an important area of investigation for moving complex image and video processing techniques "to the edge" (co-located with sensor platforms). Effective image registration will help to extract key image information close to the imaging sensor, thereby facilitating faster response and also greatly reducing the amount of data that is transmitted through the network. This latter benefit will help to reduce power and energy consumption, and improve security. In our work, we have built on the Dataflow Interchange Format (DIF) Project [5][9][10], which is a focal point of the Maryland DSPCAD Research Group at the University of Maryland. The DIF Project provides a valuable infrastructure for developing, experimenting with, and integrating computer-aided design techniques for embedded signal processing systems. We have developed new capabilities in the DIF package to demonstrate the techniques developed in this research, and provide a basis for integrating the techniques into practical design flows for optimized implementation of embedded signal processing software. We have also built on our recent work on architectures and acceleration techniques for medical image registration, which has provided a valuable foundation for the application case study thrust of this project.

The objective of this research has been to investigate: (a) an application case study on dataflow based design and implementation of high performance image registration applications; (b) improvements to software synthesis for graphics processing units (GPUs) to improve the performance of synthesized implementations; (c) library components for high performance, GPU-based implementation of image registration functions; (d) intermediate representations for optimized scheduling techniques and memory management configurations; (e) integration of the new image registration component library, synthesis tool enhancements, and intermediate representations into DIFML, which is an Extensible Markup Language (XML) format for standardized exchange of dataflow graph information; and (f) a novel software tool that enables Emulab-based experimentation using a user-friendly, formally-rooted, high level, dataflow language interface.

Approved for Public Release; Distribution Unlimited.

3. Methods, Assumptions, and Procedures

3.1. Parameterized Scheduling using Topological Patterns

For dataflow models of large-scale digital signal processing (DSP) applications, the underlying graph representations often consist of smaller sub-structures that repeat multiple times. We have demonstrated that *Topological patterns* (*TPs*) enable more concise representation and direct analysis of such substructures in the context of high level DSP specification languages and design tools [2]. Furthermore, by allowing designers to explicitly identify such repeating structures, use of TPs provides an efficient alternative to automated detection of such patterns, which entails costly searching in terms of graph-isomorphism and related forms of computation. A TP is inherently parameterized and provides a natural interface for *parameterized scheduling*, which enables efficient derivation of adaptive schedule structures that adjust symbolically in terms of design time or run-time variations.

Scheduling is a critical aspect of implementing dataflow graphs (e.g., see [1]). Parameterized schedules have been studied before (e.g., see [3][4]), and previously, production and consumption rates were key dataflow graph aspects that were used to generate parameterized schedules. In this project, we introduced a formal design method for specifying TPs and deriving parameterized schedules from such patterns based on a novel schedule model called the *scalable schedule tree* (*SST*). Our method ensures deterministic behavior of the system based on compile-time analysis of its behavior that may contain parameterizable patterns of actor and edge instantiations.

3.2. Scalable Schedule Trees

The scalable schedule tree data structure is formalized based on the *generalized schedule tree* (*GST*), which is a compact, tree-structured graphical format that can represent a variety of dataflow graph schedules [12]. In GSTs, each leaf node refers to an actor invocation, and each internal node n (called a *loop node*) is configured with an iteration count I_n for the associated sub-tree, where execution of the sub-tree rooted at n is repeated I_n times.

An SST has all of the features of a GST and additionally provides the following new features.

1. Parameterization. A node within an SST can be parameterized with a parameter set K. The semantics of how values associated with elements of K change is determined by the model of computation that is used for application specification in conjunction with the scheduling strategy that is used to derive the schedule tree. This decoupling from parameter change semantics allows the SST model to be applied to different kinds of dataflow application models and design environments.

2. Guarded execution. An SST leaf node, which encapsulates a firing (execution) of an individual actor, has an optional *guarded* attribute, which indicates that firing of the corresponding actor should be preceded by a run-time fireability (*enabling*) check. Such an enabling check determines whether or not sufficient input data is available for the actor to fire. The guarded attribute of SSTs is motivated by the enable-invoke dataflow model of computation, where guarded executions play a fundamental role.

3. Dynamic iteration counts. Loop nodes can be dynamically parameterized in terms of SST parameters, which provide capabilities for data- or mode-dependent iteration in schedules. An SST loop node L can be viewed as a parameterizable form of the constant-iteration-count loop

Approved for Public Release; Distribution Unlimited.

nodes in GSTs. An SST loop node L has an associated *iteration count evaluation function* $c_L : K \rightarrow Z+$. An implementation of c_L takes as arguments zero or more of the parameters in K. and returns a non-negative integer (zero parameters are used if the iteration count is constant). Visitation of L begins by calling c_L to determine the iteration count, and then executing the subtree of L successively a number of times equal to this count.

4. Arrayed children. In addition to leaf nodes and SST loop nodes, a third kind of internal node, called an *arrayed children node* (*ACN*), is introduced to represent schedule structures related to TPs. An ACN z has an associated array $childern_z$ which represents an ordered list of candidate children nodes during any execution of the SST subtree rooted at z. For simplicity, we assume that $childern_z$ is a one-dimensional array, but the associated formulations can easily be extended to handle multi-dimensional arrays of candidate children. The array $childern_z$ has a positive integer size $size_z$, which gives the number of elements in the array. It is assumed that the array is indexed starting at 0. Each element in $childern_z$ represents a schedule tree leaf node (i.e., an encapsulation of an actor in the enclosing dataflow graph), an SST loop node, or another SST --- i.e., a "nested" SST. An ACN z also has three functions associated with it, which we denote as $cinit_z$, $cstep_z$, and $climit_z$, that determine how $childern_z$ is traversed during a given execution of the enclosing subtree. These functions take as arguments pre-specified subsets of the parameters of z, and return, respectively, a non-negative, positive, and non-negative integer. One or more of these functions can be constant-valued - dependence on parameter settings is not essential but rather a feature that is provided for enhanced flexibility.

When an ACN z is visited during traversal (execution) of the enclosing schedule tree, the following sequence of steps, called the *SST traversal process*, is carried out. (1) The parameter settings for z are updated by applying the evaluation function f_p for each parameter $p \in P_z$. (2) The values of $cinit_z$, $cstep_z$, and $climit_z$ are evaluated in terms of the updated parameter settings. These values are stored in temporary variables, which we denote as `I`, `s`, and `L`, respectively. (3) The computation outlined by the pseudocode shown below is carried out, where `A` represents the array $children_z$; `count` represents the iteration count evaluation function of the associated SST loop node; and `K` represents the set of parameters for the enclosing SST.

```
for (i = I; i <= L; i += s) {
  if A[i] is a leaf node {
    execute the actor encapsulated by A[i]
  } else if A[i] is an SST loop node {
    Z = count(K)
    execute the loop node subtree Z times
  } else { // A[i] is a nested SST
    recursively apply the SST traversal process to A[i]
  }
}
```

3.3. Improved GPU-targeted Synthesis Tool

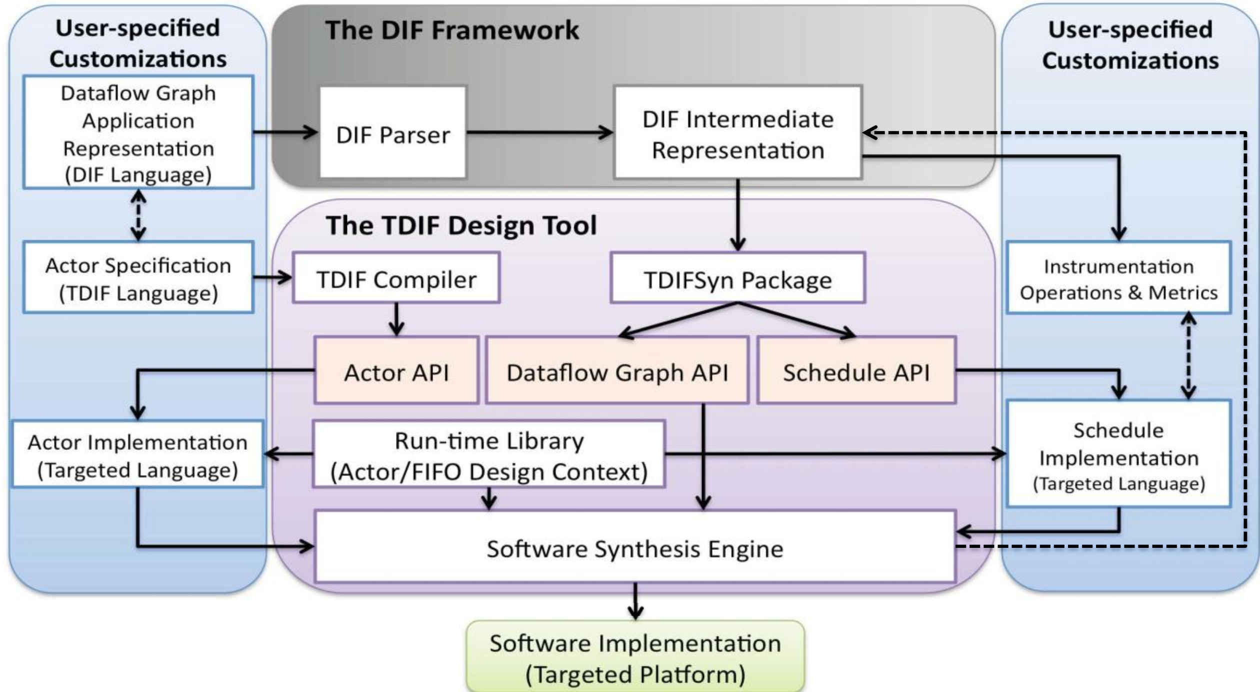

Figure 1: TDIF design flow.

Figure 1 shows the design flow using the targeted dataflow interchange format (TDIF) [5]. By following this methodology, the designer can focus on design, implementation and optimization for dataflow actors and experiment with alternative task scheduling strategies and instrumentation techniques for the targeted platforms based on programming interfaces that are automatically generated from the TDIF tool. These automatically-generated interfaces provide well-defined, structured design templates for the designer to follow in order to generate dataflow-based actors that are formally integrated into the overall synthesis tool. The TDIF environment currently supports C- and GPU-based implementations (i.e., for Central Processing Unit [CPU] and GPU platforms). The GPU-based capabilities of TDIF are currently oriented towards NVIDIA GPUs, based on the Compute Unified Device Architecture (CUDA) programming framework [11]. Since CUDA is a C-like programming language, CUDA can be viewed as a variant of C with NVIDIA extensions and certain restrictions, a C- or CUDA-based actor can be implemented as an abstract data type (ADT) to enable efficient and convenient reuse of the actor across arbitrary applications. In typical C implementations, ADT components include header files to represent definitions that are exported to application developers and implementation files that contain implementation-specific definitions.

We implemented a new plug-in to the DIF framework that extends *the DIF language* (*TDL*) to incorporate support for TPs and allows designers to construct SSTs for schedules associated with dataflow graphs that are specified in TDL. This plug-in integrates the SST formulations as a new internal representation format and associated set of manipulations within the DIF framework. TPs that are currently supported by TDL and defined as *pattern keywords* in the language include `chain`, `ring`, `merge`, `broadcast`, `parallel`, and `butterfly`. We have also developed an SST plug-in from which SSTs can be specified programmatically using graph construction Application Programming Interfaces (APIs) associated with the SST internal representation. For details on formal definitions for topological patterns in DIF and APIs defined for the SST plug-in for constructing SSTs, we refer the reader to [18] and [19], respectively.

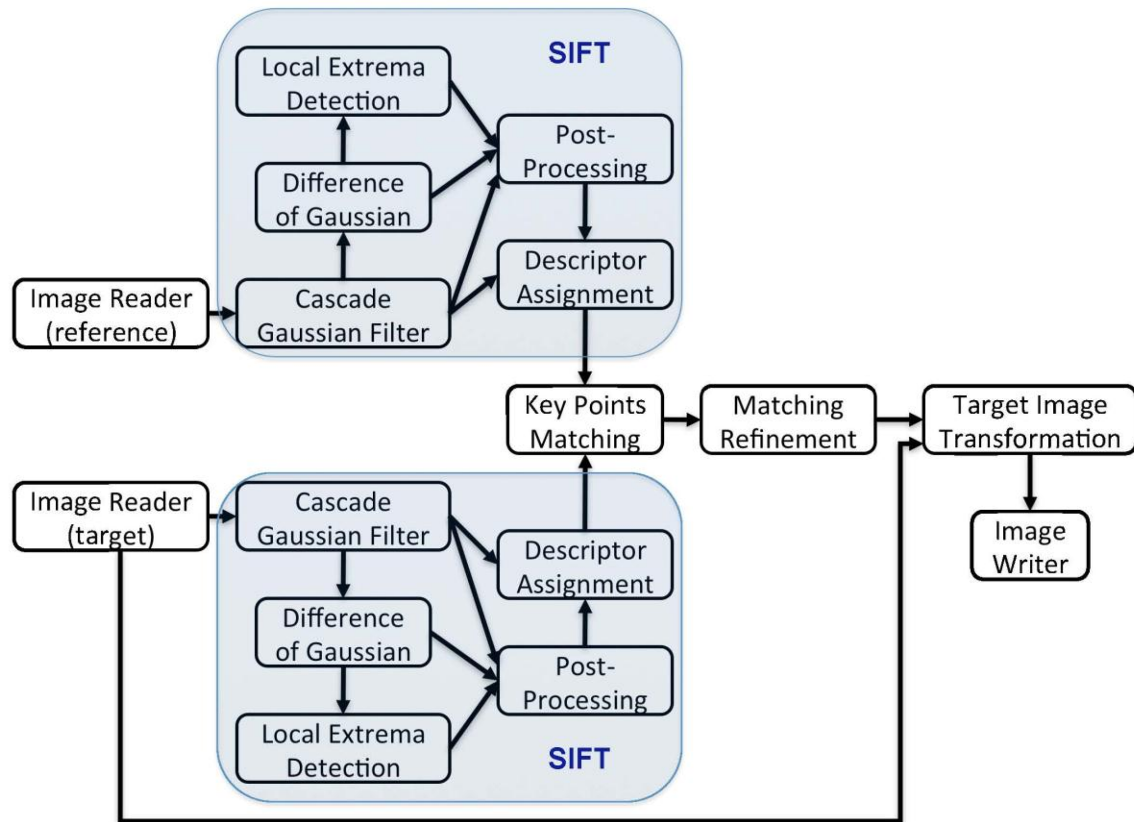

Figure 2: Design flow of the targeted image registration application.

3.4. Library Components and Application Example

3.4.1. Application: Image Registration

In this project, we used an image registration application to demonstrate our TDIF-based design and synthesis approach. Image registration is a process of geometrically aligning two or more images of the same scene so that they can be overlaid [6]. Here, one of the images is referred to as the *reference* image and the second image is referred to as the *target* image. Image registration algorithms can be classified into two types: *feature-based* and *intensity-based*. In feature-based algorithms, image features, such as points, lines, and contours, need to be identified and matched between the target and reference images. In intensity-based algorithms, intensity patterns are compared using correlation metrics.

Once corresponding features or intensity patterns have been found, a transformation method is applied to align the target image with the reference image. Generally, there are two types of transformation algorithms: *rigid transformation* and *non-rigid transformation*. Rigid transformation only consists of and rotation, translation scaling, while non-rigid transformation allows locally warping the target image.

We model a non-rigid image registration application based on the *Scale Invariant Feature Transform* (*SIFT*) [7] algorithm using dataflow graphs. **Figure 2** shows the design flow for such an image registration system in terms of a dataflow graph. The overall system is composed of

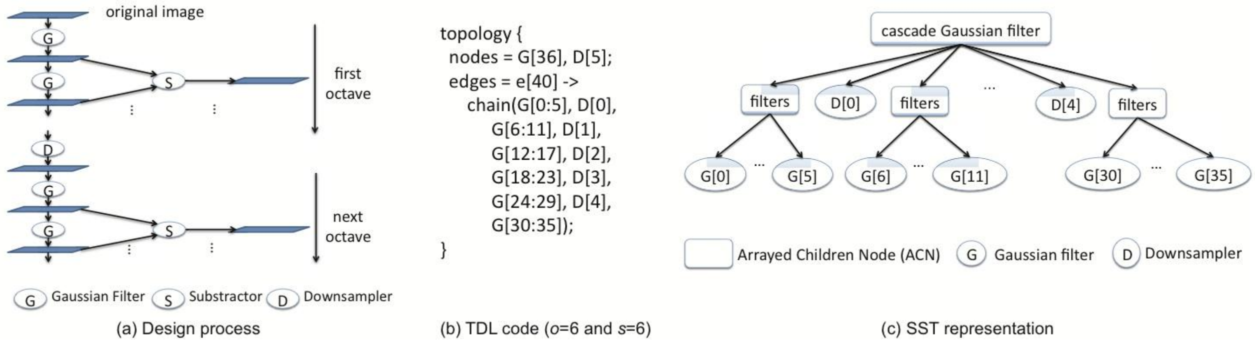

Figure 3: Cascade Gaussian Filtering.

subsystems for SIFT, the *Key Points Matching* technique, *Matching Refinement*, and *Target Image Transformation*. The SIFT algorithm is a method to extract scale and rotation invariant features from images. It can be used to perform feature matching between images that are taken from different views of the same scene. In our dataflow-based design, as shown in **Figure 2**, the SIFT algorithm is divided into five actors: *Cascade Gaussian Filtering*, *Difference of Gaussian*, *Local Extrema Detection*, *Post Processing*, and *Descriptor Assignment*. We implemented these actors using C for checking functional correctness. Here, parallelism can be achieved in the computations of *Cascade Gaussian Filtering*, *Difference of Gaussian*, and *Local Extrema Detection*. Therefore, in addition to implementing these actors using C, we also implemented the parallelizable actors in SIFT using CUDA for performance acceleration.

For the computation of key points matching as shown, in **Figure 2**, the input of the associated actor includes a descriptor array, produced from the SIFT algorithm, for the referenced image and a descriptor array, also produced from the SIFT algorithm, for the target image. This computation is annotated with a parameter $\tau > 1$ for the matching threshold. A key point i in a descriptor *D1* is matched to a key point j in a descriptor *D2* only if the Euclidean distance d_{ij} between i and j multiplied by the matching threshold is not greater than the Euclidean distance of i in *D1* to all other key points in *D2*. The output of this actor is an array that contains the matching information, i.e., matched index pairs from *D1* and *D2*.

Since key points matching may generate incorrect matches between the reference image and the target image, a refinement step is needed in order to eliminate such incorrect matches. For the computation of matching refinement, we applied the *Random Sample Consensus* (*RANSAC*) algorithm for this refinement step [8]. RANSAC is an iterative method to estimate parameters of a mathematical model from a set of observed data consisting of both inliers and outliers. In our case, inliers are correct matches and outliers are incorrect matches.

As shown in **Figure 2**, the target image transformation takes inputs from the refined matching result and the target image and produces the resulting registered image. For the computation of the target image transformation, we divide the rigid transformation of the image into three basic components: *translation*, *rotation*, and *scaling*.

TDIF specifications and associated C and CUDA implementations of the targeted image registration application are provided in the project deliverables. Evaluation results of performance are provided and discussed in Section 4.1.1.

To demonstrate our methods and associated new SST plug-in for representation of and code generation from schedules for dataflow graphs that employ TPs , we use the *Cascade Gaussian Filtering* (*CGF*) subsystem in the SIFT algorithm.

The CGF subsystem contains a number of Gaussian filters with different standard deviations. These filters produce a series of Gaussian filtered images. CGF is a relevant case

study for experimenting with TPs and SSTs because it can be modeled naturally in terms of parameterized topologies. As shown in **Figure 3**(a), CGF can be modeled as a dataflow graph consisting of actors that perform Gaussian filtering and downsampling computations. These computations can be divided into a set of *o* groups, such that each group involves *s* filtering steps. Both *o* and *s* are parameters that can be configured by the designer (e.g., to explore trade-offs between processing complexity and image processing accuracy).

In the CGF process illustrated in **Figure 3**(a), the original image is convolved with the first filter. The filtered image is saved and then convolved with the next filter, and so on. After one group of filtering operations is carried out, *s* different blurred Gaussian images are labeled as a separate octave. The next step is to downsample the last image of the previous octave by a factor of two. This process, as shown in **Figure 3**(a), repeats until *o* octaves of images are produced.

The TP underlying the CGF application is a chain (linear arrangement of actors), which can be specified in TDL. **Figure 3**(b) shows the TDL specification with $o=6$ and $s=6$. Here, an array of 40 edges is instantiated by connecting 41 specified nodes (six groups of six nodes each that are interleaved with five individual nodes) in a chain.

In this CGF example, since both *o* and *s* are parameters that can be configured, one can naturally derive a nested SST as shown in **Figure 3**(c). Such a representation provides a formal, target-language-independent model of schedule structure that can be applied to coordinate execution for this subsystem in a manner that is parameterized across two dimensions.

As shown in **Figure 3**(c)., the `cascade Gaussian filter` ACN has 11 children nodes, which include 6 nested ACNs, each labeled as `filter`, and 5 `downsampler` actors encapsulated as leaf nodes, which are labeled as `D[0]`, `D[1]`, …, `D[4]`. Each of these leaf nodes represents an encapsulation of a `downsampler` actor in the CGF application. Each internal node labeled `filter` is an ACN that contains 6 children nodes, where each of these children nodes represents an encapsulation of a `Gaussian filter` actor in the application.

TP specifications using DIF for the targeted image registration application and an example using our SST plug-in for the cascade Gaussian filtering application are provided in the project deliverables. Evaluation results of coding efficiency are provided and discussed in Section 4.1.2.

3.5. Emulab Software Tool

Dataflow-based modeling is typically not applied to networking aspects of networked signal processing applications such as the ones developed in Emulab. Network simulations involve link conditions and data protocols that are usually not represented using dataflow techniques. Network/application co-simulators address the issue of simulating the network conditions and the application at each node. However, most co-simulators today do not utilize dataflow-based modeling of the application (i.e., the intra-node functionality). As the range of network and distributed applications expands, it becomes increasingly important to develop methods to simulate the intra-node network conditions together with the dataflow models at the node level. Such a method would provide complete system analysis of networked signal processing applications without giving up the benefits of dataflow-based design practices at the level of individual nodes.

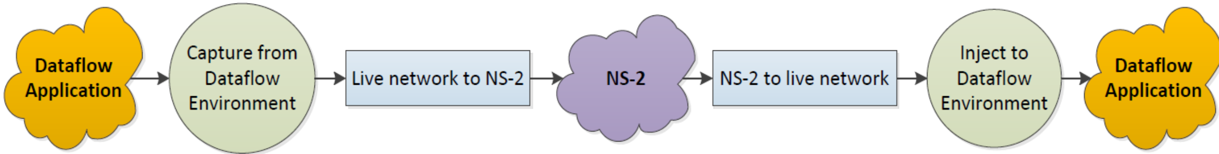

Figure 4: Illustration of the interaction between dataflow applications and network simulations in NT-Sim.

To bridge the gap described above, we have developed a co-simulation tool called *NT-SIM* (*NS-2–TDIF Simulation Environment*) that combines TDIF with the popular Network Simulator (ns-2) to provide novel capabilities for experimentation with networked signal processing systems. NT-SIM is a flexible environment that allows designers to completely simulate systems at both the node and network levels. Dataflow-based design tools are available to assist in the development of layered sensing applications and other kinds of signal processing applications for which dataflow models can be applied to derive efficient placement and scheduling solutions. At the same time, ns-2 allows for detailed analysis of network properties and their effect on node information sharing. This allows designers to understand and validate the operation of network nodes as well as their interactions in the network.

Figure 4 illustrates the execution order and interactions among components in the NT-SIM framework. Application behavior is specified based on dataflow modeling principles using the TDIF framework. To interface with the end system dataflow simulation and traffic generation for the network, the network behavior and protocols used by the nodes are defined by Object Tool Command Language (OTcl) scripts, and simulated by the NSE (NS Internal Emulator) framework.

In NT-SIM, special dataflow actors called IAs (Interface Actors) are developed to allow the sending and receiving of information between NSE and TDIF. In contrast to conventional dataflow actors, which represent functional components from the application specification, IAs are responsible for traffic generation from TDIF-based modeling subsystems, and injection of this generated traffic into the NSE framework. IAs are also responsible for time synchronization between the cooperating TDIF- and NSE-based simulation environments. This collection of IAs in a TDIF-based dataflow subsystem makes the subsystem appear as a single node within an enclosing ns-2 network topology.

The architecture of NT-SIM is designed to preserve the dataflow principles provided by the TDIF environment throughout all TDIF-based subsystems, including the interactions that occur at the interfaces of these subsystems (i.e., at the IAs). The designer is responsible for specifying the distribution of actors to the nodes in the network graph. In the NT-SIM framework, the designer develops the system in a hierarchical manner: actor design using TDIF, dataflow graph design at each network node using DIF, and network graph design using ns-2. The First-in-first-out (FIFO) communication channels in DIF act as bridges between actors in the dataflow graph. Correspondingly, the IAs act as bridges between dataflow graphs that are placed on different network nodes. In NT-SIM, dataflow subsystems can be suspended (e.g., as they wait for data) and resumed arbitrary numbers of times while the overall network is being simulated, thus allowing for simulation of complex and tightly-coupled feedback behaviors across the network.

Thus, NT-SIM provides designers with a hierarchical, modular process for modeling and experimenting with networked signal processing systems. NT-SIM also provides a useful target for incorporating additional levels of automation in the design and simulation processes. For example, protocol configurations and associated implementation details can be determined and

Approved for Public Release; Distribution Unlimited.

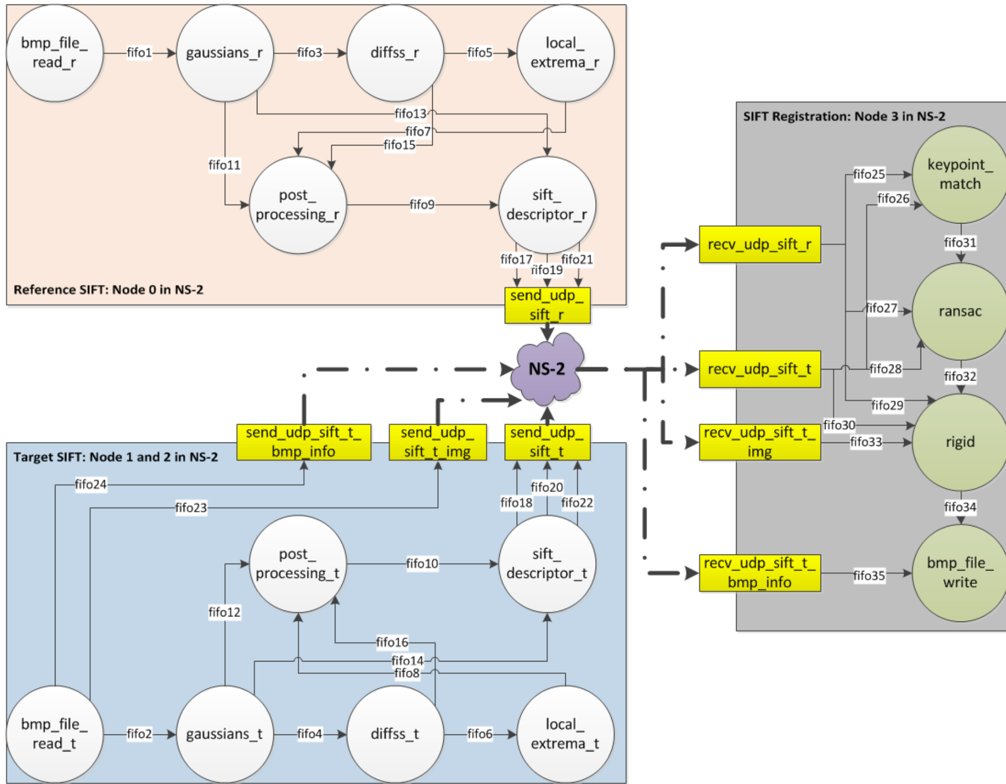

Figure 5: A dataflow graph model of SIFT-based feature detection and image registration across a network.

optimized automatically by incorporating associated IA synthesis capabilities within the TDIF synthesis engine.

The processes of design and experimentation using NT-SIM are demonstrated more concretely in the next section.

3.5.1. NT-SIM Case Study: Visual Sensor Network

We demonstrate the utility of NT-SIM with a case study of simulating a visual sensor network designed to perform image registration on different views of the same object.

Visual sensor networks (VSNs) are comprised of groups of networked visual sensors with image capture, computation, and wireless communication capabilities. To maximize the effectiveness of a VSN, collaboration among the sensors can take place with the exchange or fusing of visual information from similar or different perspectives of an area [14]. This allows the information to be used in tracking, panoramas, and registration.

Each sensor node in a VSN has to fulfill application requirements while running under constraints involving memory, performance, data rates, and energy [15]. By distributing actors appropriately across the network, more processing-intensive tasks can be performed at one or more stationary systems that are connected to power sources, while simpler tasks are handled by the sensor nodes. This allows energy on the sensor nodes to be conserved while the computationally-intensive task of image registration is carried out, and also helps to improve the performance of image registration by allowing use of more powerful (less power constrained) platforms for the registration tasks.

In this case study, we experiment with this approach of heterogeneous computing and distribution-based optimization of energy and performance for the SIFT application in a VSN. This experimentation is carried out through mapping of the dataflow graphs for distributed signal processing onto separate network nodes, configuration of IAs in TDIF for appropriate communication among the nodes, and simulation using NT-SIM. **Figure 5** shows a dataflow graph model of the SIFT algorithm being applied across a network. Here, the SIFT algorithm is used to register two images with different views of the same object.

Each of the actors in the SIFT algorithm is modeled using the TDIF environment. For this purpose, the SIFT algorithm is broken into smaller procedural units to be modeled with actors. At this level of NT-SIM, the actors are not assigned to any particular nodes in a network. The focus at the actor design level of NT-SIM is to create actors that are represented by the TDIF language. In this phase of the design process, designers specify the target language of each actor, along with the inputs, outputs, required parameters, and possible execution modes for the actor. The TDIF file for the SIFT descriptor actor, which passes the SIFT descriptor to the keypoints matching, RANSAC, and rigid transformation actors, is shown below. Here, we show the SIFT descriptor actor specified as a CUDA-targeted actor for GPU-based implementation:

```
module CUDA sift_descriptor_r

output output1 sift token
output output2 sift token
output output3 sift token

input input1 oframes
input input2 gss

mode init
mode exe
```

As another example, TDIF code is shown below for an actor that sends an image from the actor representing the capture of the target image to the network simulated by ns-2. For simplicity and clarity in the illustration, we design the network to follow the User Datagram Protocol (UDP). As a result, such an image-sending actor takes in the address and port number as character-string parameters, and these parameters are employed by the actor in addition to any inputs coming from other actors in the enclosing dataflow graph subsystem.

```
module C send_udp_sift_t_img

input input image image token*

param send addr char*
param send port char*

mode init
mode send
```

In NT-SIM, the application that runs on each network node is represented by a specification in the DIF language. To optimize the energy and performance of the SIFT VSN, actors are split onto different network nodes depending on their roles in the overall application graph. In this case study, actors are distributed across network nodes depending on whether they perform feature detection or image registration. This results in multiple dataflow graph subsystems with each subsystem corresponding to a single network node. Each of these subsystems can be specified using a DIF file that defines the actors as vertices and the connections between them as edges in the associated dataflow graph.

The current version of NT-SIM systematically integrates designer-provided tests and schedules into the overall network simulation, and automates the execution of this simulation across the entire network. Thus, NT-SIM bridges the gap between network- and dataflow-graph-level simulation in networked signal processing systems, and provides novel capabilities into which existing and newly developed dataflow scheduling techniques can be integrated to further enhance simulation automation and design space exploration.

When using NT-SIM, the designer creates a Tool Command Language (Tcl) script that models the network topology on NSE to simulate the network. In order to use NSE on ns-2, the `RealTime` *scheduler has to be used with the simulator. Nodes are declared along with the* network objects and agents. When using the UDP protocol, each of the network objects has to declare the Internet Protocol (IP) address and port number in the script. These network objects are attached to their corresponding agents. Afterwards, the connections between nodes can be defined, along with the bandwidth, delay, and queue behavior for each connection. Each agent is attached to a node. If the nodes share a common link, then the agents are also connected. Afterwards, NSE can be run. **Figure 6** illustrates the network topology used in our SIFT VSN case study.

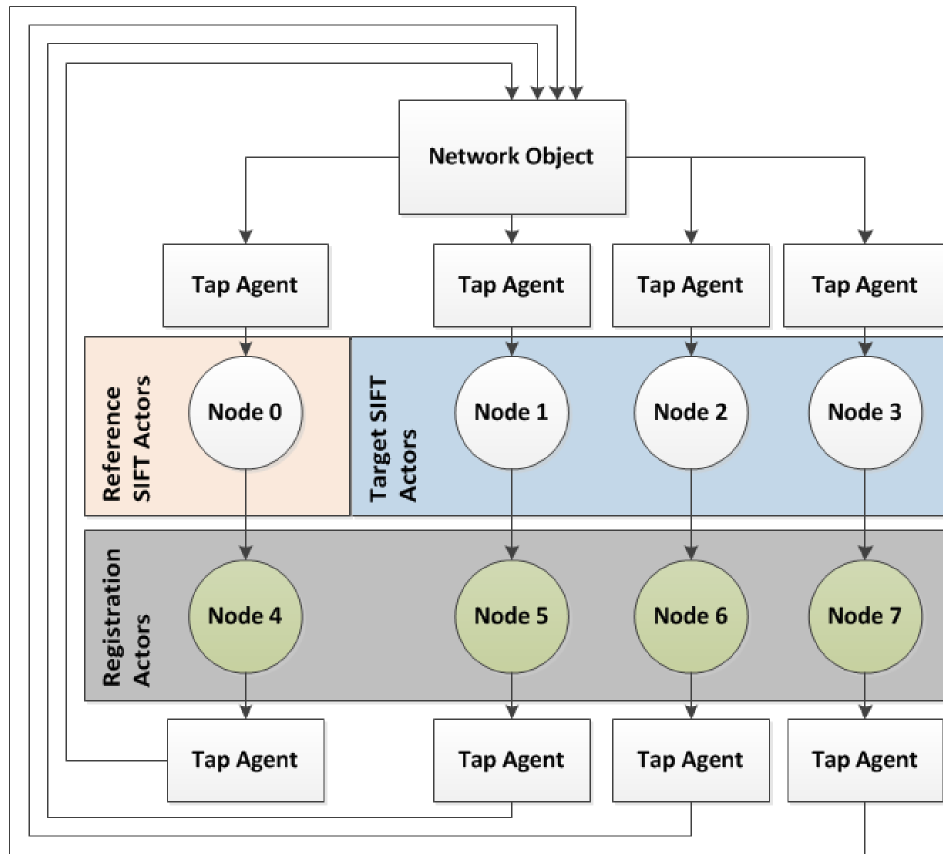

Figure 6: The topology represented by the Tcl script for the SIFT sensor network.

After the actors, dataflow graph subsystems (the portions of the dataflow graph that are mapped onto individual network nodes), and the network have been specified, the overall system can be simulated using NT-SIM. The Tcl script for the network is run using NSE. This allows network connections to be made between the TDIF and ns-2 environments. Separate test and DIF files are required for each VSN node. After the executables have been generated for each VSN node, they can be run --- concurrently with simulation of the resulting network traffic --- to send and receive data to and from NSE, respectively.

3.6. Improvements to DIFML

DIFML is a software package developed under our previous contract that provides an XML-based format for exchanging information between DIF and other tools and languages, and more generally, between arbitrary pairs of dataflow environments.

Table 1: Performance comparison between CPU-targeted and GPU-targeted actors.

Actors	CPU (seconds)	GPU (seconds)	Speedup
Cascade Gaussian filter	11.896	0.416	28.60
Difference of Gaussian	0.584	0.012	48.67
Target image transformation	0.614	0.017	36.12

Table 2: Performance comparison for the overall application with and without GPU acceleration.

CPU (seconds)	GPU (seconds)	Speedup
55.575	30.523	1.82

In this project, we used a Java code coverage tool called EMMA to analyze and provide feedback for enhancing the rigor of tests that have been created for the DIFML package. Intuitively, code coverage reports the percentage of source code components that are exercised by one or more tests in a given test suite. EMMA is a free code coverage tool that can measure code coverage results and report for a Java program [17]. Based on the results reported by EMMA, we enhanced testing for parts of the DIFML package whose code coverage results were found to be under a target threshold of 90%. As a result, our DIFML package is now validated with a test suite having at least 90% overall code coverage, which is generally considered a high level of testing rigor.

Table 3: Experiments for comparison with GPU peak performance.

Actor name	Execution time (milliseconds)	GFlops	Comparison to GPU peak performance (%)
Cascade Gaussian Filtering	13	45.19	6.3
Difference of Gaussian	0.512	152	21.3
Target Image Transformation	0.988	89.6	12.5

In the project deliverables, we provide an upgraded DIFML package with assocaited new tests. We also provide example code for the new library components involved in our targeted image registration application.

App	w/o TP	w/ TP
CGF	81	3
JPEG encoder	37	9
FFT (N=8)	32	2

(a) Comparison for TDL specifications

Top-level DIF specification	5n+e+6
TDIF specification	5n
Building SST	16
Actor development	c
Total	**10n+e+22+c**

(b) Designer-written code in TDIF

Top-level C file	9n+6
Functional declaration	56n
Scheduling APIs	22n
Scheduling file header	2n+5
Scheduling	41n
Actor development	c
Total	**130n+11+c**

(c) Implementations generated by TDIF

Figure 7: LOC evaluation results.

4. Results and Discussion

4.1. Case Study: Image Registration

4.1.1. Evaluation for Performance Acceleration

Table shows a performance comparison for the CPU-targeted actors (implemented using C) and GPU-targeted actors (implemented using CUDA) in the targeted image registration application. **Table** shows a performance comparison between two versions of the overall application --- in one version all of the actors are CPU-targeted, and in the other version, the most computationally intensive actors are GPU-targeted. As shown in **Table** and **Table** , the CUDA implement- ations have superior performance compared to the corresponding C implementations for these experiments. However, the application-level speedups, while still significant, are consistently less than the corresponding actor-level speedups. We believe that this is due to factors such as context switch overhead and communication cost for memory movement, which are associated with overall schedule coordination in the application implementations. The input for these experiments is a 1200x900 gray-scale bitmap image, and the implementations are executed on a 3 Gigahertz (GHz) PC with an Intel CPU, 4 Gigabyte (GB) Random Access Memory (RAM), and an NVIDIA GTX260 GPU.

In addition to the real-time performance comparisons that have been shown above for the GPU-targeted and non-GPU-targeted actor implementations of our targeted image registration application, we have also calculated peak performance values for the GPU-targeted actor implementations in terms of Giga Floating Point Operations Per Second (GFLOPS). Our GFLOPS calculation for an actor is based on the number of floating point operations that will be launched in the CUDA kernel for the actor divided by the execution time of the CUDA kernel. In this experiment, we measured the execution time and manually counted the number of floating point operations implemented in the CUDA kernels. The results for GFLOPS are provided in **Table** . The actors were implemented on an NVIDIA GTX260 GPU, which provides 715 GFLOPS as peak performance.

Approved for Public Release; Distribution Unlimited.

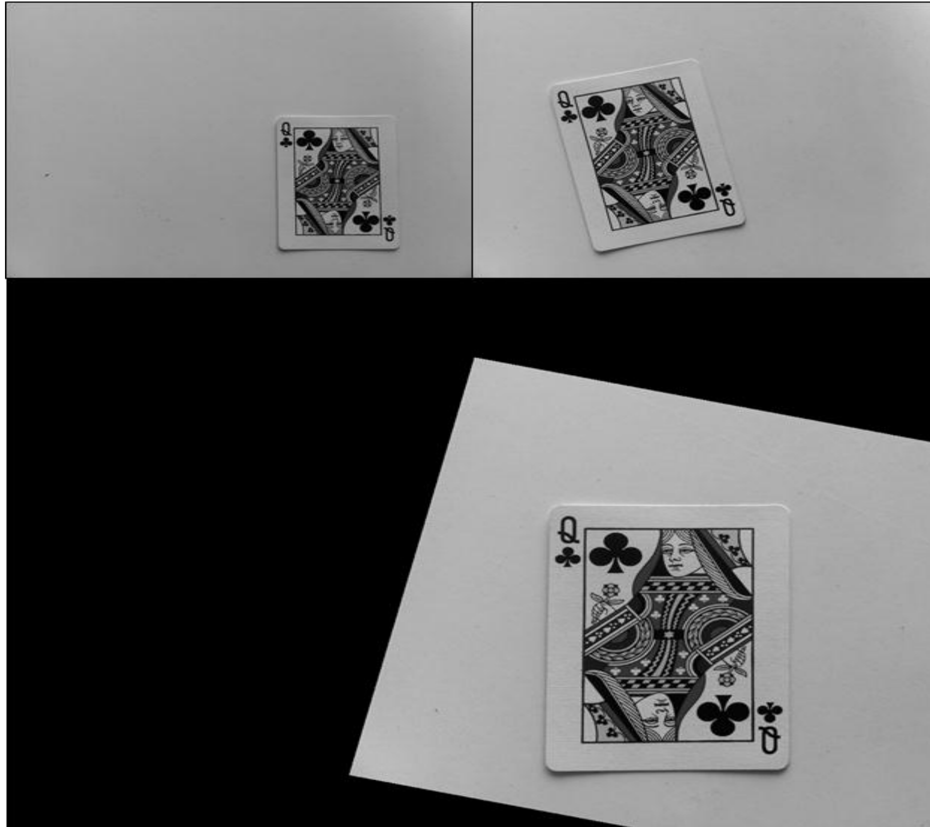

Figure 8: (Clockwise from top left) Reference image, target image, and registered image from the simulated SIFT VSN.

4.1.2. Evaluation in Terms of Coding Efficiency

We also apply an evaluation metric called the *lines of code* (*LOC*), which is the number of lines of code required for an application. We use this LOC metric to help quantify the benefits of the concise and scalable representation of DSP applications using TPs. Unless otherwise specified, the LOC cost refers to code that the designer needs to manually provide (e.g., in contrast to code that is automatically generated or reused from some other part of an implementation). We apply this metric on various applications, including the CGF application, that are specified with and without use of TPs.

We first compare LOC evaluation results, as shown in **Figure 7**(a), for different applications by using TDL with and without the support of TPs. For the specifications in this comparison, each node and edge declaration occupies a separate line of code. We also compare the LOC cost of CGF implementation that uses code generation and the LOC cost of the generated code in the TDIF environment. This gives a comparison of the complexity of the complete implementation generated using TDIF compared to the complexity of the code that the designer has to write and maintain as source code.

Figure 7(b) summarizes the LOC costs for different implementation components for the CGF application when code generation is used --- i.e., these are the costs for the designer-written code that can be viewed as input to the TDIF toolset. These costs are listed as functions of the numbers of dataflow graph actors n and edges e in the scalable application, and the total LOC costs c in the designer-written component of the actor implementations.

On the other hand, **Figure 7**(c) shows the LOC costs of the complete generated implementation --- i.e., the generated code together with the designer-written TDIF input code that is used directly (without translation) in the implementation. In the CGF application, the underlying TP is a chain, and the number of edges is of the same order as the number of nodes. Thus, comparing the LOC listings in **Figure 7**(b) and **Figure 7**(c), we see that as the number of nodes *n* in the application is increased, the ratio of the designer-written LOC cost to the complete implementation LOC cost decreases. This helps to quantify the utility of the TDIF tool in terms of LOC costs as a function of graph complexity. This comparison incorporates the use of TPs, which help to reduce the LOC cost for the top-level DIF specification.

4.2. Case Study: Visual Sensor Network

The SIFT sensor network is simulated on a 3GHz PC with two Intel Xeon CPUs, 3GB RAM, and an NVIDIA GTX260 GPU. The `gcc version 3.4.4` and `nvcc version 3.2` compilers are used in the back end of the implementation process. The functional accuracy of NT-SIM was verified through simulation of the SIFT VSN case study. End systems (network nodes) representing reference and target image sensors that can perform feature detection were supplied with only the reference and target image shown in **Figure 8**. Functional accuracy was validated by the match between the produced, registered image and a ground-truth, registered image provided by the simulation of the single-node SIFT algorithm.

5. Conclusion

In this project, we developed and delivered improved software tools and application examples for demonstrating layered sensing and signal processing systems on high performance embedded processors such as GPUs. We created GPU-enhanced dataflow components and applied our DIF/TDIF tool to demonstrate an application case study on high performance image registration applications. We also integrated the targeted image registration application into our DIFML package, which provides an XML format for standardized exchange of dataflow graph information. We introduced a formal design method for specifying topological patterns for signal processing applications and deriving parameterized schedules from such patterns based on a novel intermediate schedule representation called the scalable schedule tree (SST). We also developed a novel software tool called the NS-2–TDIF simulation environment (NT-SIM), which enables Emulab-based experimentation for networked signal processing systems. In the project deliverables, we have included the developments in TDIF, NT-SIM, and DIFML that have been supported in this project, as well as the developments in DIF with the TP plug-in and associated SST functionality demonstrated for a selected application example based on cascade Gaussian filtering.

6. Publications

The following is a list of publications that were produced as outcomes of this project.

[1] Z. Zhou, C. Shen, W. Plishker, and S. S. Bhattacharyya. Dataflow-based, cross-platform design flow for DSP applications. In A. Sangiovanni-Vincentelli, H. Zeng, M. Di Natale, and P. Marwedel, editors, Embedded Systems Development: From Functional Models to Implementations. Springer, 2013. To appear.

[2] C. Shen, S. Wu, N. Sane, H. Wu, W. Plishker, and S. S. Bhattacharyya. Design and synthesis for multimedia systems using the targeted dataflow interchange format. IEEE Transactions on Multimedia, 14(3):630-640, June 2012.

[3] L. Wang, C. Shen, G. Seetharaman, K. Palaniappan, and S. S. Bhattacharyya. Multidimensional dataflow graph modeling and mapping for efficient GPU implementation. In Proceedings of the IEEE Workshop on Signal Processing Systems, Québec City, Canada, October 2012.

[4] L. Wang, C.-C. Shen, S. Wu, and S. S. Bhattacharyya. Parameterized scheduling of topological patterns in signal processing dataflow graphs. Journal of Signal Processing Systems, pages 1-12, 2012. DOI:10.1007/s11265-012-0719-x.

[5] S. Won. A networked dataflow simulation environment for signal processing and data mining applications. Master's thesis, Department of Electrical and Computer Engineering, University of Maryland, College Park, 2012.

[6] S. Won, C. Shen, and S. S. Bhattacharyya. NT-SIM: A co-simulator for networked signal processing applications. In Proceedings of the European Signal Processing Conference, Bucharest, Romania, August 2012.

[7] S. Wu, C. Shen, N. Sane, K. Davis, and S. Bhattacharyya. Parameterized scheduling for signal processing systems using topological patterns. In Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, pages 1561-1564, Kyoto, Japan, March 2012.

[8] N. Sane. Rapid Prototyping of High Performance Signal Processing Applications. PhD thesis, Department of Electrical and Computer Engineering, University of Maryland, College Park, 2011.

[9] N. Sane, H. Kee, G. Seetharaman, and S. S. Bhattacharyya. Topological patterns for scalable representation and analysis of dataflow graphs. Journal of Signal Processing Systems, 65(2):229-244, 2011.

[10] C. Shen, H. Wu, N. Sane, W. Plishker, and S. S. Bhattacharyya. A design tool for efficient mapping of multimedia applications onto heterogeneous platforms. In Proceedings of the IEEE International Conference on Multimedia and Expo, Barcelona, Spain, July 2011. 6 pages in online proceedings.

[11] S. Wu. Representation and scheduling of scalable dataflow graph topologies. Master's thesis, Department of Electrical and Computer Engineering, University of Maryland, College Park, 2011.

[12] N. Sane, H. Kee, G. Seetharaman, and S. S. Bhattacharyya. Scalable representation of dataflow graph structures using topological patterns. In Proceedings of the IEEE Workshop on Signal Processing Systems, pages 13-18, San Francisco Bay Area, USA, October 2010.

Approved for Public Release; Distribution Unlimited.

7. References

[1] S. S. Bhattacharyya, E. Deprettere, R. Leupers, and J. Takala, Eds., *Handbook of Signal Processing Systems*. Springer, 2010.

[2] N. Sane, H. Kee, G. Seetharaman, and S. S. Bhattacharyya, “Scalable representation of dataflow graph structures using topological patterns,” in *Proceedings of the IEEE Workshop on Signal Processing Systems*, October 2010.

[3] B. Bhattacharya and S. S. Bhattacharyya, “Parameterized dataflow modeling for DSP systems,” *IEEE Transactions on Signal Processing*, October 2001.

[4] M. Ko, C. Zissulescu, S. Puthenpurayil, S. S. Bhattacharyya, B. Kienhuis, and E. Deprettere, “Parameterized looped schedules for compact representation of execution sequences in DSP hardware and software implementation,” *IEEE Transactions on Signal Processing*, June 2007.

[5] C. Shen, H.Wu, N. Sane,W. Plishker, and S. S. Bhattacharyya, “A design tool for efficient mapping of multimedia applications onto heterogeneous platforms,” in *Proceedings of the IEEE International Conference on Multimedia and Expo*, July 2011.

[6] B. Zitova and J. Flusser. Image registration methods: a survey. *Image and Vision Computing*, 21:977–1000, 2003.

[7] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, pages 91–110, 2004.

[8] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, June 1981.

[9] C. Hsu, M. Ko, and S. S. Bhattacharyya, “Software synthesis from the dataflow interchange format,” in *Proceedings of the International Workshop on Software and Compilers for Embedded Systems*, Dallas, Texas, September 2005, pp. 37–49.

[10] W. Plishker, N. Sane, M. Kiemb, K. Anand, and S. S. Bhattacharyya, “Functional DIF for rapid prototyping,” in *Proceedings of the International Symposium on Rapid System Prototyping*, Monterey, California, June 2008, pp. 17–23.

[11] *NVIDIA CUDA Compute Unified Device Architecture - Programming Guide*, 2007. [Online]. Available: http://developer.download.nvidia.com/compute/cuda/1_0/NVIDIA_CUDA_Programming_Guide_1.0.pdf

[12] M. Ko, C. Zissulescu, S. Puthenpurayil, S. S. Bhattacharyya, B. Kienhuis, and E. Deprettere, “Parameterized looped schedules for compact representation of execution sequences in DSP hardware and software implementation,” *IEEE Transactions on Signal Processing*, vol. 55, no. 6, pp. 3126–3138, June 2007.

[13] S. S. Bhattacharyya, S. Kedilaya, W. Plishker, N. Sane, C. Shen, and G. Zaki, “The DSPCAD integrative command line environment: Introduction to DICE version 1,” Institute for Advanced Computer Studies, University of Maryland at College Park, Tech. Rep. UMIACS-TR-2009-13, August 2009.

[14] K.Fall and K.Varadhan, “The ns Manual (formerly ns Notes and Documentation)”, November 2011.

[15] Y. Bai and H. Qi, “Feature-based image comparison for semantic neighbor selection in resource-constrained visual sensor networks,” *EURASIP Journal on Image and Video Processing*, 2010.

[16] I. F. Akyildiz, T.Melodia, and K. R. Chowdhury, “Wireless multimedia sensor networks: Applications and testbeds,” *Proceedings of the IEEE*, vol. 96, no. 10, pp. 1588–1605, October 2008.
[17] V. Roubtsov. EMMA Reference Manual, 2006.
[18] N. Sane, H. Kee, G. Seetharaman, and S. S. Bhattacharyya. Topological patterns for scalable representation and analysis of dataflow graphs. *Journal of Signal Processing Systems*, 65(2):229-244, 2011.
[19] S. Wu. *Representation and scheduling of scalable dataflow graph topologies*. Master's thesis, Department of Electrical and Computer Engineering, University of Maryland, College Park, 2011.

Approved for Public Release; Distribution Unlimited.

A. Appendix–Project Deliverables

A.1. Introduction to Deliverable Organization

The project deliverables are stored as sub-packages in the `afrl-dspcad-cete-installation` package and delivered as a compressed file called `afrl-dspcad-cete-installation.tar.gz`. The sub-packages in the project deliverables are 1) DIF with the topological patterns plug-in, 2) the TDIF plug-in to DIF, 3) the DIFML plug-in to DIF, 4) the SST plug-in to DIF, and 5) demo examples for items 1-4, which are labeled as `dif-demo`, `tdif-demo`, `difml-demo`, and `sst-demo`, respectively.

The `afrl-dspcad-cete-installation` package is an *IDICE* package. IDICE provides tutorial/instructional extensions to the DICE (the *DSPCAD Integrative Command Line Environment*) package [13] for streamlined and configurable use of DICE. DICE is a package of utilities that facilitates efficient management of software projects. Use of IDICE and DICE provides a unified framework for introducing and applying important software engineering methods and practices, such as script-based automation, design for cross-platform operation, unit testing, and incremental project development.

The TDIF sub-package stores the APIs for the delivered GPU software modules, run-time libraries, and GPU-targeted software synthesis tools. The associated deliverables are stored in the following directory:

```
afrl-dspcad-cete-installation/idice/idice-set/libs/tdifgen
```

The DIFML sub-package stores the DIFML software. The associated deliverables are stored in the following directory:

```
afrl-dspcad-cete-installation/idice/idice-set/libs/difmlgen
```

DIF with the topological patterns plug-in and SST plug-in are stored in the following directory:

```
afrl-dspcad-cete-installation/idice/idice-set/libs
```

The `dif-demo`, `tdif-demo`, `difml-demo`, and `sst-demo` directories store demonstration examples that are implemented using the DIF pakcage with the topological pattern plug-in, the TDIF package for the targeted image registration application, the TDIF package with the assocaited NT-SIM plug-in, the DIFML package, and the SST package, respectively. The associated deliverables are stored in the following directory:

```
afrl-dspcad-cete-installation/idice/idice-set-adm/dist
```

All deliverables in the `afrl-dspcad-cete-installation` package are built under the Ubuntu Linux platform with NVIDIA's GPU, CUDA, and ns-2 enabled.

A.2. Instructions for Deliverable Installation and Startup

To install the deliverables using IDICE, follow these steps:

1. Copy the file `afrl-dspcad-cete-installation.tar.gz` to the user's home directory.
2. Extract the `afrl-dspcad-cete-installation` directory from the `tar.gz` archive in which it is packaged.

   For example:

   ```
   cd ~
   tar xvf afrl-dspcad-cete-installation.tar.gz
   ```

3. Create the `afrl-dspcad-cete-user` directory in the user's home directory.

   For example:

   ```
   cd ~
   mkdir afrl-dspcad-cete-user
   ```

4. Create a directory named `startup` in the afrl-dspcad-cete-user directory.

   For example:

   ```
   cd ~/afrl-dspcad-cete-user

   mkdir startup
   ```

5. Copy the `IDICE` startup file from the `afrl-dspcad-cete-installation` directory to the `afrl-dspcad-cete-user/startup` directory.

   That is (all on a single line of input):

   ```
   cp ~/afrl-dspcad-cete-installation/idice/idice-set-
   adm/setup/idice_set_startup
   ~/afrl-dspcad-cete-user/startup
   ```

To start up the software using `IDICE`, follow these steps:

1. Start a `bash` shell.
2. `cd` to the `afrl-dspcad-cete-user` directory --- e.g., run:

   ```
   cd ~/afrl-dspcad-cete-user
   ```

3. Run

   ```
   bash -norc

   source startup/idice_set_startup
   ```

A.3. Instructions for the Demonstrations

After the startup of IDICE, a set of utilities is provided as a companion to DICE for convenient interaction with the project deliverable environment. These include:

idxget <file>

This allows the user to get a local copy (e.g., a C file example) of a distributed file from the deliverable set.

idxupdate <directory>

This allows the user to get a local copy (e.g., a C file example) of a distributed directory from the deliverable set.

idxlist <no arguments>

This lists the current set of distributed files and directories along with their associated modification dates.

Note that idxget and idxupdate overwrite any previous version of the file/directory in your current working directory. Distributed directories are generally distributed as tar.gz archives, so use idxupdate for entries that show up (with idxlist) with .tar.gz endings, and use idxget for other (non-archive) entries.

To learn about more useful DICE commands, please refer to [13].

The demo examples --- dif-demo.tar.gz, tdif-demo.tar.gz, sst-demo.tar.gz and difml-demo.tar.gz --- are stored in the distribution directory of the afrl-dspcad-cete-installation package. These archives can be listed by using the idxlist command, and the user can use the idxupdate command to copy the demo examples into the afrl-dscpad-cete-user directory by following these steps:

```
cd ~/afrl-dspcad-cete-user
idxupdate tdif-demo
idxupdate dif-demo
idxupdate sst-demo
idxupdate difml-demo
```

A.3.1. Demo for Image Registration using TDIF

The demonstrated image registration application using TDIF is stored in tdif-demo/ir. In both tdif-demo/ir/src and tdif-demo/ir/test, a makeme script contains commands to performs all necessary compilation steps that are needed for the test. Here, three compilation steps are needed. The first step is to parse actor-specific TDIF files and generate APIs for the corresponding actors. Actor designers can then provide the associated implementation code (in C or CUDA) based on the provided APIs. The second step is to parse the DIF files, which are specified in the DIF language, extract the overall dataflow graph structure, generate a corresponding top-level C file that implements the input dataflow graph, and generate a header file for designers to implement schedulers. The third step is to use the NVCC (NVIDIA CUDA Compiler) to compile all of the implementation files (i.e., *.c and *.cu files) and generate the final executables. That is, by linking with the TDIF run-time library, the associated actor object code (compiled from C or CUDA), and scheduler object code, the generated top-level C file can be

compiled using NVIDIA's NVCC compiler. The resulting executable can then be run on the targeted NVIDIA GPU platform. To perform all the compilation steps in `tdif-demo/ir/src`, use the following command:

```
makeme
```

In `tdif-demo/test`, a script called `runme` contains a command to run the resulting executable, which takes an input reference and target BMP files, respectively, and produces a registered bitmap image file (BMP) after performing image registration tasks, as described in Section 3.4.1, on the NVIDIA GPU platform.

A.3.2. Demo for SIFT Visual Sensor Network using NT-SIM

The demonstrated visual sensor network simulation (described in Section 3.5.1), which simulates the targeted, SIFT-based image registration application using NT-SIM, is stored in `tdif-demo/vsn`. In `tdif-demo/vsn/src`, a `makeme` script contains commands to performs the three compilation steps for all the required dataflow components specified using TDIF. To run a network simulation using NT-SIM in this demo, four terminal sessions (bash sessions) need to be used, and each session should have `IDICE` enabled. Then users need to follow the steps below in each session:

1. In terminal session 1, go to the `tdif-demo/vsn/test/test-network` and use NSE to launch a localhost network specified using ns-2 script. That is,
   ```
   nse network.tcl
   ```
2. In terminal session 2, go to the `tdif-demo/vsn/test/test-receiver`. Execute the `makeme` and `runme` scripts, repectively, to compile the code needed for the receiver node and listen to the designated port for the arrival of packets. The receiver node will wait util it receives packets from the refenerce node and target node, respectively, to start processing and generate a registered image.
3. In terminal session 3, go to the `tdif-demo/vsn/test/test-sender-reference`. Execute the `makeme` and `runme` scripts, repectively, to compile the code needed for the reference node and transmit the packets for the processed reference image to the localhost with the designated port.
4. In terminal session 4, go to the `tdif-demo/vsn/test/test-sender-target`. Execute the `makeme` and `runme` scripts, repectively, to compile the code needed for the target node and transmit the packets for the processed target image to the localhost with designated port.
5. A registered image will be produced on the receiver node (i.e., from terminal session 2).

A.3.3. Demo for Image Registration using Topological Patterns

The demonstrated image registration application using DIF that incorporates support for topological patterns is stored in `dif-demo/tp/ir`. In `dif-demo/tp/ir/test`, users should execute the `makeme` script to compile a driver program that takes a DIF file as input. This input DIF file specifies the targeted image registration application using topological patterns. After executing the `makeme` script, users should execute the `runme` script to construct a dataflow graph for the targeted image registration application, and produce the graphical result

(i.e., a rendering of the constructed dataflow graph), which is stored in Portable Network Graphics (PNG) format.

A.3.4. Demo for cascade Gaussian filtering using SST plug-in

The demonstrated scalable schedule tree (SST) representation for scheduling the cascade Gaussian filtering (CGF) application is stored in `sst-demo/cgf`. In `sst-demo/cgf/test`, users should execute the `makeme` script to compile a driver program that constructs an SST for CGF as shown in **Figure 3**(c) using the SST plug-in. After executing the `makeme` script, users should execute the `runme` script to construct an SST and produce relevant information about the constructed schedule tree to standard output.

A.3.5. Demo for Image Registration using DIFML

The `difml-demo` directory contains demo examples, including the targeted image registation application, that use the `DIFML` tool to transform files between the DIF and DIFML formats. In each example, a `runme` script contains a command to run the DIFML parser, which takes an input file in either DIF or DIFML format and produces an output file in either DIFML or DIF format, respectively. After running each `runme` script, the corresponding output file will be generated automatically.

The contents of the `tdif-demo` directory includes the following subdirectories.

- `src`: stores the source code that has been written for the demo, i.e., the Gaussian filtering application.
- `bin`: stores executable programs resulting from the source code after compilation.
- `test`: stores tests for the demo.

LIST OF ACRONYMS

ACN	Arrayed Children Node
ADT	Abstract Data Type
API	Application Programming Interface
BMP	Bitmap Image File
CGF	Cascade Gaussian Filtering
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DICE	DSPCAD Integrative Command Line Environment
DIF	Dataflow Interchange Format
DIFML	Dataflow Interchange Format Markup Language
DSP	Digital Signal Processing
FIFO	First-in-first-out
GB	Gigabyte
GFLOPS	Giga Floating Point Operations Per Second.
GHz	Gigahertz
GPU	Graphics Processing Unit
GST	Generalized Schedule Tree
IDICE	Instructional DSPCAD Integrative Command Line Environment
JPEG	Joint Photographic Experts Group

Approved for Public Release; Distribution Unlimited.

LOC	Lines of Code
NSE	NS Internal Emulator
NT-SIM	NS-2–TDIF simulation environment
NVCC	NVIDIA CUDA Compiler
PNG	Portable Network Graphics
RAM	Random Access Memory
RANSAC	Random Sample Consensus
SIFT	Scale-Invariant Feature Transform
SST	Scalable Schedule Tree
TP	Topological Pattern
TDIF	Targeted DIF
TDIFSyn	TDIF Synthesis
TDL	The DIF Language
TDP	The DIF Package
UDP	User Datagram Protocol
XML	Extensible Markup Language