# 내장형 스트리밍 어플리케이션을 위한 매개변수 데이터플로우 모델 기반의 C++ 확장

## (A C++ Extension based on a Parameterized Dataflow Model for Embedded Streaming Applications)

최 윤 서 † Yuan Lin ††
(Yoonseo Choi) (Yuan Lin)

**요 약** 내장형 신호처리 시스템의 상당 수는 스트리밍(streaming) 어플리케이션의 특성을 지니고 있다. 데이터플로우(dataflow) 계산모델을 이용하면 스트리밍 프로그래밍 패러다임을 손쉽게 표현할 수 있다. 데이터플로우 계산모델에서는 프로그램의 병렬성이 드러나므로 멀티코어를 위한 병렬 프로그램으로의 컴파일 과정 또한 용이해진다. 우리는 내장형 신호처리 시스템의 스트리밍 특성을 데이터플로우 계산모델에 기반하여 표현하기 위한 언어 확장으로서 SPEX(Signal Processing Extension)을 제안하고자 한다. SPEX는 기존의 명령형언어(imperative language)상에 스트리밍 프로그램밍 패러다임을 표현할 수 있게 한다. SPEX 언어 확장은 매개변수 데이터플로우 계산모델(parameterized dataflow)에 기반하고 있으며, 이를 위해 몇몇의 키워드를 기존의 C++ 언어 더하는 방식으로 이루어져 있다. 본 논문에서는 하나의 필터 내에서의 스트리밍 계산 특성 및 필터 간의 스트리밍 데이터 전달을 표현하는 SPEX의 기능에 초점을 맞추고자 한다.

**키워드** : 스트리밍 어플리케이션, 언어 확장, 매개변수 데이터플로우, 내장형 신호처리 시스템

***Abstract*** Many DSP systems are streaming applications in which streams of data constantly flow through a set of filters. Dataflow programming paradigm is one of effective methods for representing these streaming applications. Dataflow programming model explicitly exposes parallelisms within an application, which helps compiling of the application onto a multicore platform. We propose SPEX(Signal Processing Extension), a language extension to a standard imperative language based on the parameterized dataflow model. Parameterized dataflow model is a kind of dataflow model that can express a modest fashion of dynamism contrary to the synchronous dataflow that can represent only static dataflow. SPEX facilitates characterizing an application written in conventional imperative languages such C/C++ as a streaming application. SPEX is comprised of a few keywords augmented to the conventional C++ syntax for representing dataflow paradigm. SPEX also restricts the syntax and semantics of C++ in order to fit the program within a certain streaming programming category. In this paper, we focus on the capability of SPEX in representing streaming computations within filters and streaming communications among filters.

**Key words** : Streaming application, language extension, parameterized dataflow, embedded DSP system

· 이 논문은 2007년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(KRF-2007-357-D00200)

† 정 회 원 : University of Michigan EECS 박사후 연구원
yoonseo.choi@gmail.com
†† 정 회 원 : Cavium Networks 엔지니어
nauynil@gmail.com

논문접수 : 2008년 11월 28일
심사완료 : 2009년 2월 10일



정보과학회논문지 : 컴퓨팅의 실제 및 레터 제15권 제4호(2009.4)

## 1. 서 론

멀티미디어 컨텐츠가 더 빠르고 복잡해짐에 따라, 디지털신호처리 시스템도 매우 복잡하고 다양해지고 있다. 이러한 디지털신호처리 시스템의 효과적인 수행을 위해 프로그램 가능한 디지털신호처리 아키텍처들이 개발되고 있으며, 최근에는 더 많은 계산량의 처리를 위해 멀티코어 아키텍처들도 나타나고 있는 추세이다. 이러한 디지털신호처리 아키텍처는 프로그래밍이 가능하지만, 아키텍처의 복잡성과, 디지털신호처리 시스템의 특성으로 인해, 효과적인 소프트웨어 개발에는 많은 시간과 비용이 소모되고 있는 실정이다.

오디오, 비디오, 디지털신호처리, 네트워크 어플리케이션에 이용되는 커널들은 끊임없이 입력 데이터를 처리하여 출력 데이터를 생성하는 스트리밍 특성을 지닌다. 예를 들면, MPEG 비디오의 경우, 비트 스트림 및 인코드된 매크로 블록 데이타가, zigzag, quantization, IDCT, motion vector decode 등, 여러 개의 독립적인 필터들을 지나게 된다. 필터들은 데이터 입출력을 통하여, 즉, 하나의 필터의 출력 데이터가 다음 번 필터의 입력 데이터가 되는 방식으로 서로 데이터를 주고 받으면서, 연결된 형태를 가진다. 따라서, 스트리밍 어플리케이션에서 각각의 필터는 입력받은 데이를 처리하는 과정에 있어서는 다른 필터들과는 상관없이 독립적이다. 또한, 필터들간의 커뮤니케이션이 데이터의 입출력으로 프로그램 상에 직접적으로 드러난다.

이러한 특성들로 인해서, 스트리밍 어플리케이션은 데이터플로우 모델로 잘 표현될 수 있다. 가장 간단한 데이터플로우 모델로 SDF(Synchronous Data Flow)가 있다. SDF는 각각의 액터(필터)가 생성하고 소모하는 토큰의 개수가 즉 데이터의 양이 상수이고 컴파일 타임에 이미 알려져 있는 특성을 지닌다. 이러한 제약으로 인해, 동작을 미리 예측할 수 있고, 컴파일 타임에 액터들을 스케줄링 할 수 있으며, 각종 최적화가 가능한 특성을 지닌다. 그림 1은 간단한 SDF의 예를 보여준다. 그림에서 노드 A, B, C는 액터들을 나타내고, 스트리밍 어플리케이션에 대입해서 생각해 보면, 세 개의 필터들을 나타낸다. 에지에 양 끝에 표시된 숫자는 해당 액터가 한 번 수행될 때 마다 생성되고 소모되는 토큰의 개수, 즉 데이터의 양을 나타낸다. 그림 1에서는 노드 A가 한 번 수행될 때 마다 세 개의 토큰을 생성해 내고, 반면 노드 B 가 한 번 수행될 때마다 두 개의 토큰을 소모하므로, 노드 A가 두 번, 노드 B가 세 번, 노드 C가 세 번 수행되는 식의 스케줄이 가능하다.

그러나, 컴파일 타임에 모든 동작이 결정되는 SDF만으로 스트리밍 어플리케이션을 모두 표현하기에는 부

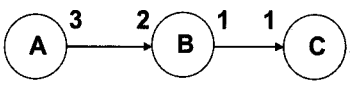



그림 1 간단한 SDF의 예

족할 수 있다. 실제적인 어플리케이션에서는 동적으로 데이터플로우의 동작을 제어할 수 있는 기능이 필요하다. 이를 위해, SDF의 구성 요소들을 매개변수화 하여 런타임시에 셋팅해주는 매개변수 데이터플로우, PSDF (Parameterized Synchronous DataFlow)가 제안되어 왔다[1]. PSDF를 이용하면, 각 액터가 생성하고 소모하는 토큰의 개수, 액터들 간의 연결 구성 등을 정해진 범위 내에서 매개변수를 통하여 동적으로 제어할 수 있다. 예를 들면, 그림 2의 (a)에서는 노드의 입력 토큰의 개수와 출력 토큰의 개수가 주어진 집합의 원소 중 하나가 될 수 있음을 나타내고 있다. 입출력 토큰의 개수는 각각 input_rate와 output_rate 라는 매개변수의 형태로 표현되어 있다. 또한, (b)에서는 if_cond에 따라서 데이터의 흐름이 true 노드를 따를 수도, false 노드를 따를 수도 있음을 보여준다. 마지막으로 (c)에서는, 노드의 개수가 split_factor에 따라서 주어진 집합 안에서 변할 수 있음을 보여준다. 그림 2의 Input_rate, output_rate, if_cond, split_factor는 모두 런타임시의 데이터에 따라서 주어진 범위 내에서 값이 변할 수 있는 매개 변수들이다.

데이터플로우의 스케줄링을 용이하게 하고, 메모리 사용량을 제한하기 위해서, PSDF의 매개 변수들은 일반적으로 일정한 범위 내의 값을 가진다. 또한, 런타임시에 아무 때나 매개 변수의 값이 변하기보다는, 특정한 초기화의 단계에만 매개 변수의 값이 설정되는 제한적인 동적 수행 방식을 따른다. 초기화에 설정된 매개 변수의 값에 따라서 데이터플로우가 결정되고, 결정된 데이터플로우가 한동안 수행되다가, 다시 다른 값으로 매개 변수가 초기화 되는 과정이 반복된다. 그림 3은 PSDF의 한 예 및 스케줄링의 예를 보여준다[1]. 초기화 단계에서만 수행되는 액터 rndInt2는 매개 변수인 factor의 값을 정해줄 뿐, 그림 2(b)의 repeat (5) times 문에 해당하는 본격적인 데이터플로우에는 관여하지 않는다. 반면에, rndInt1은 데이터플로우에 관여하여 정해진 factor 값만큼 반복된다. (rndInt1은 1개의 토큰을 생성하고 dnSmpl은 factor 개의 토큰을 소모한다.) 또한, dnSmpl은 Propagate에 의해 정해진 매개 변수 phase 값을 취하여 계산을 수행한다. SDF 및 PSDF에 관 한더 자세한 내용은 참고문헌 [1]과 [2]에서 찾아볼 수 있다.

본 논문에서는 디지털신호처리 시스템을 명령형 언어상에 스트리밍 어플리케이션으로서 표현하기 위한 언어

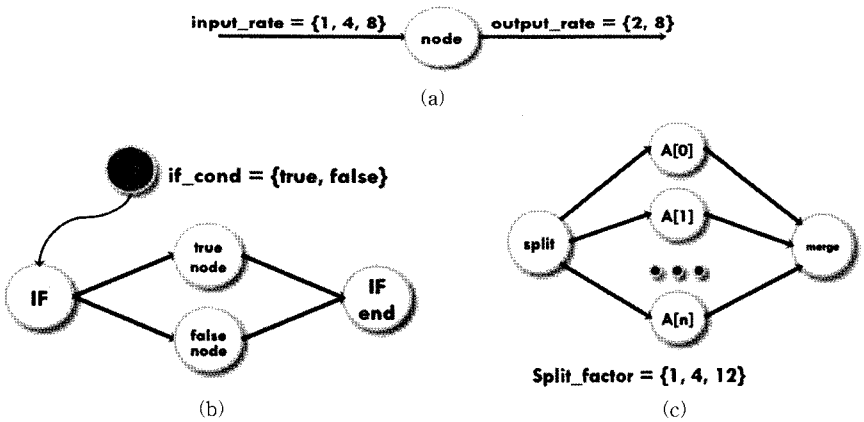



그림 2 매개변수 데이터플로우에서의 매개변수의 예

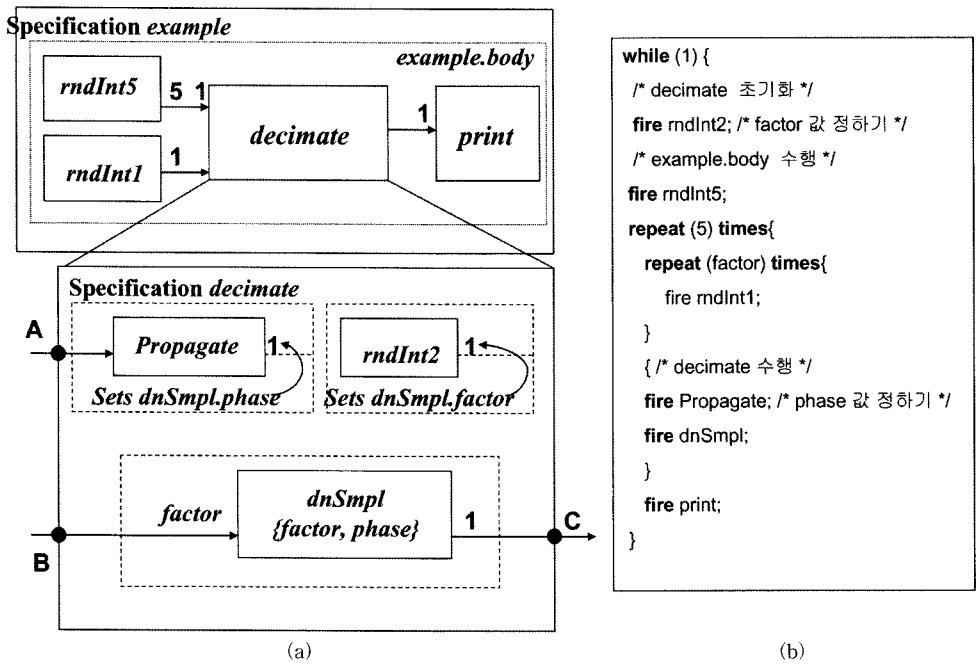



그림 3 (a) PSDF 시스템의 예: decimate 노드는 또다른 데이터플로우로 구성되어 있다. Decimate의 외부로의 인터페이스는 각각, A, B, C로 나타나 있다. (b) 이 시스템의 스케줄의 예[1]

확장인 SPEX 를 제안한다. SPEX 언어 확장은 PSDF에 기반을 두고 있으며, 디지털신호처리 시스템의 데이터 전달을 효과적으로 표현하기 위해서, PSDF 의 토큰 전달 방식을 확장하였다. SPEX의 자세한 기능에 대해서는 [3]에 서술되어 있다. 본 논문은 W-CDMA를 예로 이용하여, SPEX로 표현된 필터의 초기화-수행-최종화의 동적인 계산 방식 및, 데이터 전달을 용이하게 표현하기 위한 메모리 액터에 초점을 맞춘다. 본 논문은 영문판[4]에 더해, 오디오, 비디오 및 디지털신호처리 시스템의 스트리밍 어플리케이션으로서의 특징에 대한 소개와, 스트리밍 어플리케이션의 표현방식으로서 데이터플로우 계산 모델의 일종인 SDF 및 PSDF에 대한 개략적인 설명을 보강하였다.

### 1.1 SPEX의 특성

**매개변수 데이터플로우 계산 모델** 디지털신호처리 시스템의 스트리밍 계산 특성을 표현하기 위한 방법으로서 데이터플로우(dataflow) 계산 모델이 이용되어 왔다. 그러나 대부분의 디지털신호처리 시스템들은 데이터

의 흐름 뿐 아니라, 제어(control)의 흐름 또한 지니고 있는 것이 사실이다. 외부 환경의 변화에 대응하기 위해서는, 디지털신호처리 시스템의 수행 중간 중간에 이따금 스트리밍 데이터가 처리되는 필터들의 연결, 즉, 스트리밍 패턴이 재구성 되어야 한다. SPEX 언어 확장은, 매개변수에 의해 데이터플로우가 정의되는, 매개변수 데이터플로우(parameterized dataflow) 계산모델에 기반하고 있다. 각각의 매개변수들은 가능한 유한한 값을 가지며, 이에 의해 가능한 매개변수들의 집합이 결정되고, 이 집합들은 결국 가능한 스트리밍 계산 패턴의 구성를 나타낸다. 매개변수 데이터플로우 계산 모델에 기반하여, 디지털신호처리 프로그램 수행 중에 스트리밍 계산 패턴이 이따금씩 재구성 되는 것을 다음과 같은 삼단계 과정의 반복으로 표현할 수 있다.

단계 1. 매개변수에 상수를 할당하여, 데이터플로우를 초기화

단계 2. 초기화된 동기(synchronous) 데이터플로우 수행

단계 3. 단계 2의 계산 결과를 데이터 플로우의 상태변수에 반영하여 데이터플로우를 갱신

**직관적인 스트리밍 데이터 전달 표현 방식** 매개변수 데이터플로우 계산 모델이 스트리밍 패턴의 재구성(reconfigure)를 표현하는 데에 효과적이기는 하나, 데이터플로우 커널간에 데이터가 선입선출법(first-in-first-out, FIFO)만으로 전달된다는 제약점을 지니고 있다. 그러나 최근의 내장형 스트리밍 어플리케이션에서의 데이터 전달은 종종 복잡하고 다양하므로, 이러한 선입선출법 만으로 표현하려면, 프로그래머가 선입선출 데이터의 버퍼링을 일일히 코딩하고 관리해주어야 하는 불편함이 있다. 액터들 간의 데이터 전달을 선입선출법 이외의 방법으로도 프로그래머의 의지에 따라 자유롭게 표현할 수 있도록, 데이터의 저장 및, 접근, 액터들 간의 공유에 관여하는 메모리 액터를 제안하였다.

**명령형 언어에 기반한 언어 확장** 매개변수 계산 모델은 디지털신호처리 시스템을 모델링하기 위하여 이용되어 왔다[1]. 그러나, C나 C++와 같은 기존의 명령형 언어의 인기를 감안할 때, 프로그래머가 이러한 새로운 병행성(concurrency) 프로그래밍 모델을 바로 이용하기를 기대하는 것은 무리이다. 이와 같은 어려움을 극복하기 위해서, SPEX 언어 확장은, 프로그래머에게 이미 친숙한 C++ 언어 문법 구조에 기반하여 매개변수 계산 모델을 표현할 수 있도록 고안되었다. 또한 우리는 기존의 C++ 언어의 문법 구조 또는 의미 구조의 일부의 변경을 통해, 내장형 디지털신호처리 아키텍처로의 코드 생성을 효과적으로 이룰 수 있음을 발견하였다. 현재 SPEX는 C++의 문법 구조에 기반을 두고 있으나, 개념적으로는 어떠한 명령형 언어에도 적용될 수 있다. SPEX는 크게, 매개변수 데이터플로우를 표현하기 위해서, 새로운 몇 가지 문법구조를 제안하고, 호스트 언어(C++)의 기존의 표현능력(expressiveness)을 일부 제약한다.

다음 장에서 우리는 디지털신호처리 시스템의 수행 특성에 관한 분석 및 매개변수 데이터 플로우 모델을 이용하고자 하는 동기를 다룰 것이다. 3장에서는 우리의 SPEX에 대해서 다루고, 4장에서는 기존의 연구들에 대해서 살펴보고자 한다. 마지막으로 5장에서는 연구에 대한 결론을 언급할 것이다.

## 2. 스트리밍 디지털신호처리 시스템의 모델링

이 장에서는 첫째로, 매개변수 데이터플로우를 이용하여 스트리밍 계산을 나타내고자 하는 동기를 설명하고, 둘째로, 스트리밍 데이터 전달을 위한 목적으로 매개변수 데이터플로우의 데이터 전달 형태를 수정하고자 하는 동기를 밝힐 것이다. W-CDMA 무선통신 프로토콜의 물리적 계층(physical layer)의 이용하여, SPEX의 특성을 설명하고자 한다. 그림 4(a)는 W-CDMA의 시스템 레벨의 다이어그램을 보여준다. W-CDMA의 수신기(receiver)는 FIR, rake receiver, interleaver와 channel decoder커널들, 즉, 필터들로 이루어져 있다. 스트리밍 계산이란 데이터가 이들 커널의 입력 및 출력으로서 순서대로 연결된 커널들을 지나면서 가공되는 것을 의미한다. 또한, 스트리밍 계산 패턴의 구성이란 어떠한 종류의 커널들이 어떠한 모습으로 연결되어 있는 가를 의미한다. 이들 알고리즘 커널들은 피드포워드 파이프라인(feedforward pipeline)으로 연결되어 있다. 이 논문에서 앞으로 커널, 알고리즘 커널, 알고리즘 함수, 액터(actor) 등은 모두 같은 뜻으로 이용된다. 또한, 매개변수 데이터플로우와 PDF(Parameterized Synchronous DataFlow)는 같은 뜻을 나타낸다.

### 2.1 W-CDMA 스트리밍 계산의 모델링

**W-CDMA에서의 스트리밍 계산** 그림 4(a)의 W-CDMA에서도 알 수 있듯이, 일반적으로 스트리밍 어플리케이션에서는 커널, 즉, 알고리즘 함수들은 파이프라인과 같은 모양으로 서로 연결되어 있고, 이러한 함수들의 입력 및 출력으로서 데이터들이 함수들간에 계속적으로 전달된다. 이러한 스트리밍 어플리케이션의 함수들의 연결은, 사용자의 입력 또는 외부 환경으로부터의 변화에 대응하기 위해 때때로 재구성되곤 한다. 또한, 대부분의 디지털신호처리 시스템들은 각각의 서비스 종류에 특화된 여러 가지 수행 모드를 지원한다. 이는, 스트리밍 속도, 알고리즘 커널 및 커널의 배치의 변화를 포함한다. 예를 들면, W-CDMA는 15Kbps에서부터 2Mbps까지의 여러 가지 데이터 전송 속도를 지원한다. 음성

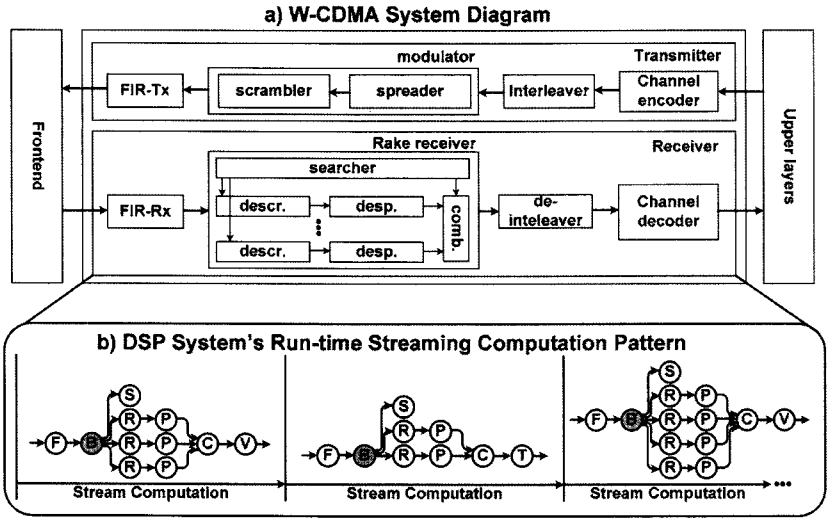



그림 4 (a) W-CDMA의 시스템 레벨의 다이어그램, (b) 런타임 스트리밍 패턴

통신을 위해서는 낮은 속도의 데이터 전송이 이용되고, 빠른 속도의 데이터 통신을 위해서는 2Mbps가 사용된다. 이러한 전송 속도 상의 차이로 인해, 다른 종류의 디지털신호처리 알고리즘 커널과 스트리밍의 재구성이 요구된다. 그림 4(b)는 스트리밍 계산 중에 주기적으로 재구성이 일어나는 상황을 보여준다. W-CDMA는 스트리밍 계산중에, 서로 다른 개수의 rake receiver를 이용할 뿐만 아니라, 서로 다른 종류의 channel decoder 알고리즘을 이용한다(그림 상에 Rake receiver는 R과 P로 나타나 있고, channel decoder 알고리즘은 T와 V로 표현되어 있다).

SDF에서는 노드(액터) 및 에지의 집합으로 표현될 수 있다(그림 1 참조). 그러나 이러한 SDF을 이용하면, 사용되는 액터의 종류 및 액터들 간의 구성이 항상 고정되어야 하므로, rake receiver의 개수가 변하는 상황 및 channel decoder 알고리즘의 종류가 변할 수 있음을 표현하기 어렵다. 뿐만 아니라, 하나의 액터가 소모하고 생성해 내는 데이터의 양이 항상 일정해야 하므로(그림 1의 에지에 나타난 숫자), W-CDMA의 데이터 전송속도가 15Kbps부터 2Mbps까지 변할 수 있음을 표현할 수 없다.

SPEX는 매개변수 데이터플로우라는 보다 동적인(dynamic) 데이터플로우를 기반으로 한다. PDF에서는 데이터플로우의 속성들이 상수가 아닌 매개변수로 나타내 진다. 매개변수는 불연속한 값을 가지는 유한한 집합, 또는 유한한 연속적인 범위를 가지는 집합으로 이루어져 있다. 대부분의 디지털신호처리 시스템들이 몇 가지 불연속한 값을 가지는 유한한 개수의 수행 모드를 지원하기 때문에, 우리는 PDF를 디지털신호처리 시스템을 표현하는데 적당하다. 디지털신호처리 시스템의 스트리밍 계산을 효과적으로 표현하기 위해서, 우리는 다음의 네 가지의 데이터플로우의 속성을 매개변수화 하여야 함을 발견하였다.

- 가변의 데이터플로우 속도: 데이터플로우 액터들의 스트림 속도는 일정한 범위 내에서 변할 수 있다.
- 조건부(conditional) 데이터플로우의 분기 조건: 분기 조건을 매개변수화 함으로써 조건부의 데이터플로우를 표현할 수 있다.
- 데이터플로우 액터의 개수: 런타임시에 이용되는 액터들의 집합 중 부분 집합이 이용될 수 있다.
- 스트리밍 크기: 각 액터의 매 회 수행 시마다 처리하는 데이터의 양이 변할 수 있다.

**런타임 수행 모델과 컴파일러 지원** 데이터플로우의 수행 스케줄은 각 액터가 어떠한 순서로 몇 회씩 어떤 하드웨어에서 수행되는지를(invocation) 표현한다. 이러한 수행 스케줄은 컴파일러에 의해 정적으로 생성될 수도 있고, 런타임 스케줄러에 의해 동적으로 생성될 수도 있다. 일반적으로 내장형시스템들은 꽉 짜인 성능 상의 제약을 지닌다. 이들은 또한 캐쉬(cache)메모리보다는 소프트웨어적으로 관리되는 스크래치패드(scratchpad) 메모리를 가지고 있다. 따라서, 런타임 시의 스케줄링 및 메모리 관리의 부담을 줄이기 위해서는 컴파일러에 의해 생성된 스케줄을 이용하는 것이 효과적이다.

SDF모델을 이용하면, 수행 스케줄을 컴파일 시에 결정할 수 있는 장점이 있다[10]. SPEX의 PDF는 매개변수를 통해 동적인 변화를 지원하지만, 일단 매개 변수

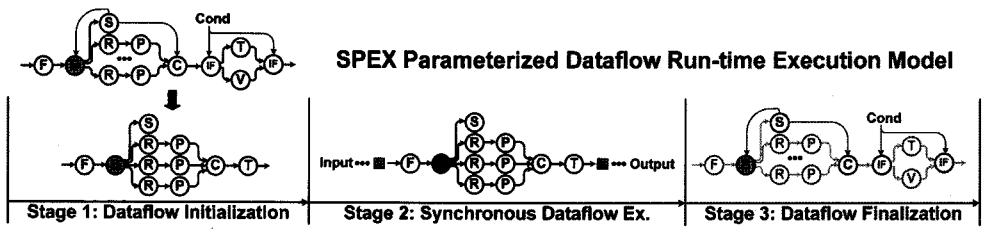



그림 5 PDF 수행 모델의 세 단계: 1) 초기화, 2) 동기 데이터플로우 수행, 3) 최종화

들이 모두 상수 값으로 셋팅되고 나면 SDF가 되는 특성을 지닌다. 이러한 특성에 따라, SPEX는 그림 5에서와같이 삼 단계로 이루어진 수행 모델을 따른다. 첫째는 초기화 단계로, 매개변수에 상수가 할당되어, 실제적으로는 PDF가 SDF가 되는 효과를 낸다. 둘째로, 데이터플로우 수행 단계에서는, 이 SDF 데이터플로우를 위해 생성된 컴파일시의 스케줄이 수행된다. 마지막, 최종화 단계에서는 이전의 수행 단계에서의 결과로부터 데이터플로우의 변수 및 상태를 갱신한다. 이 세 단계가 반복된다. 이러한 세 단계의 수행 모델은 SDF 수행 스케줄의 효율성을 유지하면서도, 초기화와 최종화를 통해서 데이터플로우를 재구성할 수 있는 유연성을 제공한다. 대부분의 디지털 신호처리 시스템은 런타임시에 아무때나 구성이 변하기 보다는, 일정 시간 동안 하나의 모드로 동작하다가, 다른 모드로 넘어가는 식으로 동작한다. 따라서, 이와 같이, 중간 중간 특정 모드에 해당하는 초기화를 거쳐 새로운 값으로 매개 변수를 셋팅하고, 그에 해당하는 동작을 한 동안 수행하다가 다음 번 모드로 넘어가기 위해 다시 매개 변수를 셋팅해 주는 초기화를 하는 방식의 수행 모델이 필요한 만틈의 동적 메커니즘을 제공하면서도, 대부분의 스케줄링이 컴파일 시에 일어나도록 하여, 런타임시의 오버헤드를 줄인다는 면에서 적당하다.

이러한 삼 단계의 수행 모델을 지원하기 위해서 SPEX의 컴파일 과정 역시 세 부분으로 나누어질 수 있다. 첫째로는 유효한 SDF 구성들을 찾아내는 단계이고, 둘째는, 각각의 SDF를 스케줄링 하는 단계이며, 마지막으로는 여러 개의 SDF 중 하나를 선택하도록 런타임 제어 코드를 생성하는 단계이다. 컴파일러는 모든 매개변수의 집합을 반복하여 탐색함으로써, 가능한 유효한 SDF 구성들을 찾아낼 수 있다. 각각의 SDF 구성들의 컴파일시에 수행 스케줄은, 기존의 SDF 스케줄링 방법론을 이용하여 찾아낼 수 있다. 마지막으로, PDF그래프의 초기화 및 최종화 단계에, 런타임 제어 코드를 삽입하게 된다.

### 2.2 W-CDMA에서의 스트리밍 데이터 전달 모델링

매개변수 데이터플로우 모델이 스트리밍 계산을 나타내는 데는 매우 효과적이지만, 상당수의 데이터신호처리 시스템들의 복잡한 데이터 전달 형태를 PDF의 일차원 선입선출법으로만 나타내기에는 어려움이 있다. 다음은 선입선출법만으로 표현하기에는 불편한 데이터 전달 형태들의 예이다.

- 다차원 스트리밍 패턴: 대다수의 디지털신호처리 시스템들은 벡터 또는 행렬에 관한 연산을 하므로 다차원의 스트리밍 패턴을 저장할 수 있는 메모리 버퍼를 필요로 한다.
- 비순차적인(non-sequential) 스트리밍 패턴: 대다수의 디지털신호처리 함수간의 데이터 전달은 엄밀한 선입선출법을 따르지 않는다. 예를 들면, 복소수 필터는 배열 안의 실수부 또는 허수부를 일정한 간격을 두고 순차적으로 접근한다(strided access). 또, interleaver는 배열의 원소를 이미 계산된 랜덤한(random)한 순서로 접근한다.
- 분리된(decoupled) 스트리밍: 상당수의 디지털신호처리 시스템은 여러 개의 서로 분리된 데이터플로우 계산으로 이루어져 있다. 이러한 분리된 데이터플로우 계산들은 메모리 버퍼들로 연결되어 있을 수 있으나, 각각 비동기적으로(asynchronously) 계산되곤 한다. 일례로, W-CDMA 수신기에서 프론트엔드(front-end)필터는 주기적으로 일정한 기한내에 동작을 마쳐야 한다. 데이터 프론트엔드 필터를 통해 15Kbps에서 2Mbps사이의 낮은 속도로 변환된 후, 벡엔드 디코더(backend decoder)를 통과한다. 그러나 백엔드 디코더는 엄밀한 실시간성 기한을 가지고 있지 않다. 이렇듯, 프론트엔드 필터와 동기적으로 작동하지 않는 디코더들이 많이 있다.
- 공유된 데이터 버퍼: 여러 개의 읽는 액터(reader)와 쓰는 액터(writer)들 간에 공유된 데이터를 가지고 있는 경우가 있다. 그러나 선입선출법에 의하면 읽고 쓰는 것에 의해 데이터가 바로 소모되고 생성된다. 즉, 선입선출법의 큐(queue)에 데이터가 들어가거나 나오게 된다. 따라서 액터들 간에 데이터가 공유되더라도 메모리를 공유하여 사용할 수 없는 단점이 있다.

기존의 연구들은 이러한 다양한 스트리밍 패턴들을 다양한 종류의 데이터플로우 계산 모델을 제안함으로써 해결하고자 했다. 예를 들어, 벡터나 배열의 스트리밍을 위해서는 다차원 데이터플로우 모델이 제안되었다[5]. 일정한 간격을 두고 순차적으로 접근되는 스트리밍 패턴을 지원하기 위해서는 순환-정적 데이터플로우(Cyclo-static dataflow) 모델이 제안되었다[6]. 그러나 이런 식의 국부적인 해결 방법론들은 특정한 데이터 전달 패턴만을 위한 것이라는 한계점이 있다. 반면에, 우리는 SPEX에서 프로그래머들이 원하는 데이터 전달 패턴을 자유롭게 구성할 수 있도록, 데이터플로우 계산 모델을 릴랙스(relax)하였다. 기존의 데이터플로우 계산 모델처럼 각각의 액터 또는 에지마다 스트리밍 패턴을 묘사하는 대신에, SPEX를 이용하면 프로그래머들이 데이터플로우 액터 및 비데이터플로우 함수들을 사용하여 복잡한 스트리밍 패턴을 표현할 수 있다. 이러한 액터들과 비데이터플로우 함수들은 데이터플로우 그래프 상에 명시적으로 연결되어 있지는 않지만, 동일한 데이터를 공유하도록 표현 될 수 있다. 데이터플로우 액터들은 장시간동안 지속적으로 일어나는 스트리밍 패턴을 표현하는 데 이용되고, 비데이터플로우 함수들은 이따금씩 데이터 접근을 제어하는 데 이용된다. 메모리 접근에 관여한다는 의미에서, 이러한 특별한 기능의 액터들은 메모리 액터라고 불리우고, 비데이터플로우 함수는 메모리 함수라고 불리운다. 이러한 메모리 액터와 메모리 함수들을 이용해서 다양한 스트리밍 데이터 전달 패턴을 표현할 수 있다. 우리의 방식의 단점은 데이터 일관성(consistency)가 문제가 있다는 점이다. 전통적인 데이터플로우 모델에서는 액터들 간에 데이터가 공유되지 않기 때문에 이러한 문제가 일어나지 않는다. 우리가 제안한 방식의 PDF모델에서는 공유된 데이터에 접근하기 위해서는 lock/unlock 메커니즘을 이용해야 하며, 데이터 일관성 문제는 궁극적으로 프로그래머의 책임이다. 그러나, 우리의 수행모델은 SDF의 컴파일시에 생성된 정해진 스케줄을 따르므로, 일반적인 멀티쓰레드 프로그램에 비해 훨씬 정적인 수행 행태를 보인다.

앞에서 언급된 네 가지 스트리밍 형태들은 모두 메모리 액터와 메모리 함수들을 사용하여 표현될 수 있다. 두 개의 읽는 액터와 한 개의 쓰는 액터를 가진W-CDMA벡터 스트리밍 버퍼의 예가 그림 6에 있다. 이 버퍼에는 네 가지 모두의 스트리밍 형태가 쓰인다. 1) 첫째로, 이 버퍼는 다차원의 스트리밍 패턴을 필요로 하는 벡터 버퍼이다. 2) 이 버퍼는 비순차적인 스트리밍 패턴을 가진다. 읽는 액터의 스트리밍 주소가 주기적으로 바뀌어야 하기 때문이다. PDF의 수행의 초기화 단계에서 읽는 주소를 설정해 주는 메모리 함수를 실행시킴으로써 이를 지원할 수 있다. 3) 이 버퍼의 쓰는 액터와 읽는 액터들 서로 다른 실시간성 기한에 따라 움직이는, 서로 분리된 액터들이다. 4) 이 버터는 두 개의 읽는 액터와 한 개의 쓰는 액터 간에 공유된 버퍼이다. 두 개의 액터들에 의해 모두 읽힌 이후에야 데이터를 버퍼에서 팝(pop) 함으로써 데이터 일관성 문제를 해결할 수 있다. 이를 위해서는 PDF수행의 최종화 단계에서야 데이터를 팝하면 된다.

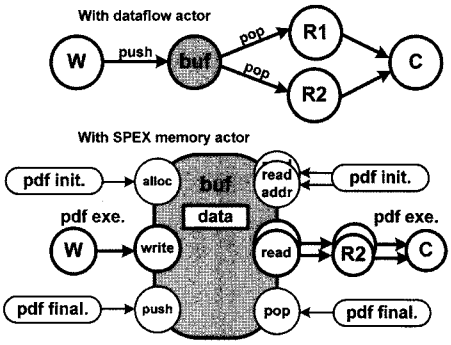



그림 6 1개의 쓰는 액터와 2개의 읽는 액터를 가진 벡터 스트림 버퍼의 예

## 3. SPEX 개요

동시성을 위한 언어 확장인 SPEX는 어떠한 명령형 언어에나 적용할 수 있지만, 본 논문에서는 범용성을 위해 C++에 적용하였다. SPEX는 크게, 매개변수 데이터플로우를 표현하기 위한 몇 가지 문법 구조와, 컴파일 편의성을 목적으로 호스트 언어(C++)의 표현능력(expressiveness)의 한계를 표현하기 위한 몇가지 문법 구조로 이루어져 있다. 이 장의 나머지는 다음과 같이 구성되어 있다. 3.1장에서 우리는 PSDF를 표현하기 위한 몇가지 새로운 구성 요소들을 소개할 것이다. 3.2장에서는 SPEX PDF 액터의 의미구조에 대해 설명하고, 마지막으로 3.3장에서는 SPEX PDF 그래프의 의미구조를 논의하겠다.

### 3.1 SPEX 구성 요소

**매개변수** param이라는 키워드로 표현되는 매개변수는 프로그래머에 의해 주어진 값들의 유한한 집합을 가진다. 매개변수들은 PDF액터 또는 그래프의 다양한 속성을 표현하는 데 이용된다. 매개변수의 문법 구조는 param<base_type> var_name 이다. 현재는 정수 타입의 매개변수만이 지원되고 있다. param<T> ::range(T min, T max)와 param<T> ::values(T val*1*, T val*2*, …, T val*n*)의 두 가지 문법으로 매개변수의 범위를 나타낼 수 있다. 첫 번째는 연속적인 범위를 나타내고, 두

번째는 불연속적인 값들의 집합을 나타낸다. 매개변수에는 사칙 연산 및 논리 연산만이 지원된다.

**채널(channel)** 채널은 데이터플로우 에지의 선입선출 큐에 해당하며, channel이라는 키워드로 표현된다. channel<base_type> chan_name 이라는 문법을 이용하여 채널을 정의하고 선언할 수 있다. 정수(integer) 및 부동 소수점 실수(floating point) 타입의 채널이 지원된다. PDF 데이터플로우 에지의 입력 및 출력 스트리밍 속도는 매개변수로서 표현될 수 있다. 구체적으로는, 채널 접근 함수에 의해서 스트리밍 속도가 정의된다. 채널 접근 함수로는, 푸쉬(push)와 팝(pop) 두 가지가 있다. 읽는 액터는 채널이 비어있으면, 채널에 읽을 데이터가 생길 때까지 기다린다. 마찬가지로 쓰는 액터는 채널이 가득차 있으면, 채널에 쓸 수 있는 공간이 생길 때까지 기다린다. 채널 변수의 크기는 컴파일러의 최적화에 의해서 결정된다.

**PDF 함수** PDF함수들은 PDF 데이터플로우 그래프의 노드인 액터 및 그래프의 기능을 표현하는데 쓰인다. pdf_node(func_name)(arg*1*, arg*2*, …)와 pdf_graph (func_name)(arg1, arg2, …)는 각각 PDF액터와 그래프 함수를 나타내는 키워드이다. PDF 함수의 인수는 PDF 액터의 입력 또는 출력 에지에 해당하는 채널 타입을 가질 수 있다. PDF 함수가 PDF 그래프를 나타낼 때도 역시 그래프 전체의 출력 및 입력이 존재하므로채널 타입의 인수를 가질 수 있다. 이렇게 데이터의 입력 및 출력을 나타내므로, PDF 함수의 인자들은 읽기-전용 또는 쓰기-전용이 되며, 일기-쓰기 모두 가능한 인자들은 지원되지 않는다. 읽기-전용 함수 인자는 C++의 pass-by-value 문법구조를 따르고 func(arg_type arg_val) 같이 쓰인다. 반면에, 쓰기-전용 함수 인자는 C++의 pass-by-reference문법구조를 따르고 func(arg_type &arg_val) 같이 쓰인다. pdf_node 함수는 채널 이외에도 매개변수를 인자로 가질 수 있다. 더 나아가서 pdf_graph 함수는 메모리 액터를 인자로 취할 수 있다. 또한, PDF함수 콜(call)은 기존의 함수 콜과는 다른 의미 구조를 가진다. 각각의 PDF 함수 콜은 명시적으로 PDF 오브젝트들을 만들어 낸다. 즉, 동일한 pdf_node 함수를 여러 번 부르게 되면, 동일한 PDF 액터들이 여러 번 복사되어 생성된다.

**SPEX 언어의 한정(restriction)** SPEX는 변수나 함수의 동적인 할당을 지원하지 않는다. 모든 변수나 함수는 정적으로 선언되어야 한다. 즉, C++의 버추얼(virtual) 함수, 동적 메모리 할당 등 동적인 언어 특성은 SPEX에서 지원되지 않는다. 디지털신호처리 시스템의 동적인 특성은 매개변수에 의해 표현될 뿐이므로, C++의 동적인 언어 특성이 도움이 되지는 않는다. 변수나 함수를 정적으로 정의하고 선언 할당하게 한정지음으로써, 런타임시의 시스템의 부담을 줄이는 보다 효율적인 코드를 생성할 수 있다.

### 3.2 SPEX PDF 액터

PDF 액터들은 C++의 클래스로 표현된다. 또한 PDF 액터 클래스는 spex_kernel과 spex_memory의 두 가지 클래스로부터 반드시 상속된 클래스여야 한다. spex_kernel는 데이터플로우의 알고리즘 액터를 정의하고, spex_memory는 2.2 장에 설명된 바 있는 메모리 액터를 정의하는 데 쓰인다. SPEX에서는 spex_kernel과 spex_memory로 부터 상속된 클래스들만이 선언될 수 있으며, 이들 클래스만이 pdf_node와 같은 PDF함수를 사용할 수 있다. pdf_node와 같은 PDF함수만으로도 PDF액터의 기능을 표현할 수 있으나, spex_kernel 및 spex_memory와 같은 클래스를 사용하면 다음과 같은 객체지향 프로그래밍의 장점을 얻을 수 있다. PDF 데이터플로우 계산에 직접적으로 연관된 알고리즘 함수 이외에도 각종 초기화 오퍼레이션, 계수(coefficient) 갱신 등 자잘하게 필요한 기능들을 클래스의 멤버 메쏘드로 포함시킬 수 있다. 즉, 디지털신호처리에 관련된 모든 기능들을 하나의 클래스 안에 그룹으로 모을 수 있는 것이다. 이러한 객체지향적 특징은 간결한 프로그래밍 인터페이스를 제공할 뿐만 아니라, 컴파일러가 디지털신호처리 아키텍처 상에 코드를 배치하는 데에도 도움을 준다.

**SPEX 커널 클래스** SPEX 커널 클래스는 spex_kernel로부터 상속된 클래스를 의미한다. SPEX 커널 클래스는 디지털신호처리 알고리즘들을 데이터플로우 액터로서 표현하는데 쓰인다. 데이터플로우 액터들은 서로 상태를 공유하지 않고, 독립적이므로, SPEX커널 클래스안에는 public 변수가 허용되지 않는다. 다만, PDF 함수들은 public 멤버로서 정의될 수 있다. PDF 함수가 아닌 함수들도 SPEX 커널 클래스에 정의될 수 있으며, 이들은 SPEX의 언어 한정 이내에서 일반적인 C++ 함수의 수행 모델을 따른다. 그림 7은 SPEX 커널 클래스로서의 FIR 필터의 구현을 보여주고 있다. 이 예에서, 라인 10에 정의되어 있는PDF 함수는 읽기-전용과 쓰기-전용의 채널을 인수로 지니고 있다. 필터의 내부적인 상태는 라인 4와 5에 정의되어 있다.

**SPEX 커널 클래스 언어 한정** 데이터플로우 계산 모델에서는 액터들 간에 공유된 변수를 허용하지 않기 때문에, 하나의 커널 클래스는 하나의 PDF 함수만을 가질 수 있다. PDF 함수는 비-PDF 함수를 부를 수 있지만, 비-PDF함수들은 PDF함수를 부를 수 없다. PDF 액터의 입출력 스트리밍 속도는 채널 변수의 푸쉬와 팝 오퍼레이션에 의해서 결정된다. 푸쉬와 팝 오퍼레이션은

```
01 template<class T, TAPS>
02 class FIR: spex_kernel {
03  private:
04   T coeff[TAPS];
05   T z[TAPS];
06   ...                    Read-only input channel    Write-only output channel
07  public:                        ↓                          ↓
08   FIR() { ... }
09   ~FIR() { ... }
10   pdf_node(run)(channel<T> in, channel<T>& out) {
11     ...
12     z[0] = in.pop();                    ← FIR's dataflow input
13     for (i = 0; i < TAPS; i++) {
14       sum += z[i] * coeff[i];
15     }
16     out.push(sum);                      ← FIR's dataflow output
17     ...
18   }
19 };
20 ...
21 void main() {
22   FIR<int, 64> fir;
23   fir.run(in_chan, out_chan);           ← Creating a PDF actor
24 };
```

그림 7 FIR 필터의 spex_kernel 클래스로의 구현의 예

PDF 함수 내에서만 불릴 수 있다. 반복루프의 바디(loop body)나 조건문에서는 푸쉬와 팝 오퍼레이션은 불릴 수 없고, 다만 푸쉬/팝에 관련된 매개변수의 값만이 바뀔 수 있다.

**SPEX 메모리 클래스** SPEX 메모리 클래스는 spex_memory로부터 상속된 클래스를 의미한다. 그림 6에서 예시된 벡터 스트리밍 버퍼의 SPEX메모리 클래스로의 구현이 그림 8에 나타나 있다. pdf_node 키워드에 의해 메모리 액터가 정의되어 있다. 이 버퍼는 두 개의 읽는 액터에 의해 공유되므로, 결국, Buffer::read를 부르는 콜러(caller)가 두 개 있다. 따라서 이 read PDF 함수는 두 개의 콜러를 구분할 수 있어야 한다. 라인 10의 out_port 키워드에 보여지듯, 이는 두 개의 서로 다른 포트에 의해 구분된다. 라인 26 및 27에 나타난대로, 각각의 읽기 포트는 하나의 읽기-전용 채널에 명시적으로 바인딩된다. 각각의 포트는 유일한 ID를 가지고 있고, 두 개의 서로 다른 포트에 의해 두 개의 서로 다른 읽는 액터가 구분된다. 전통적인 데이터플로우 모델에서는 데이터의 공유가 허용되지 않으므로, 데이터 일관성 문제가 대두되지 않는다. 반면에 SPEX의 메모리 액터들은 데이터를 공유할 수 있으므로, 데이터의 일관성을 유지하는 것은 lock 메커니즘을 통한 프로그래머의 몫이다. 이를 위해, 라인 5의 spex_mutex와 같이 mutex 변수가 지원된다.

**SPEX 메모리 클래스 언어 한정** SPEX 메모리 클래스는 pdf_node 타입의 PDF 함수만을 가진다. 여러 개의 액터에 의해 데이터가 공유될 수 있으므로, 여러 개의 pdf_node 타입의 PDF 함수를 가질 수 있다. 또한 입력-포트 및 출력-포트가 정의되어야 한다. 선언된 입력-포트 및 출력-포트는 이 메모리 액터에 연결된 쓰는

```
01 template<class T>
02 class Buffer: spex_memory {
03  private:
04   T array[1000];
05   spex_mutex buf_lock;  ← Use mutex to preserve data consistency
06   int read_addr[2];
07   ...
08  public:
09   in_port writer;       ← One writer and two
10   out_port reader[2];     readers for this buffer
11   ...
12   pdf_node(read)(channel<int>& out) {
13     out_port port1 = out.src_port;
14     int reader_id = get_port_id(port1.id);
15     T dat = array[read_addr[reader_id]];
16     out.push(dat);
17     buf_lock.lock();
18     read_addr[reader_id]++;
19     buf_lock.unlock();
20   }
21 };
22 void main() {
23   Buffer<int> buf;
24   channel<int> chan1;
25   channel<int> chan2;
26   chan1.src_bind(buf.reader[0]);
27   chan2.src_bind(buf.reader[1]);
28 }
```

There are two copies of this memory actor for the two readers. Using each reader's unique port ID to distinguish between the readers

그림 8 1개의 쓰는 액터와 2개의 읽는 액터를 가지는 벡터 스트림 버퍼의 spex_memory로의 구현의 예

액터와 읽는 액터의 개수를 나타낸다. 그림 8의 라인 9와 10에서 보듯, 입력-포트와 출력-포트는 각각 in_port와 out_port라는 키워드로 표현된다. 입력 및 출력-포트의 개수는 정적으로 정의되어야 한다. 또한 하나의 포트는 명시적으로 반드시 하나의 읽기-전용 채널 또는 쓰기-전용 채널에 바인딩되어야 한다.

### 3.3 SPEX PDF 그래프

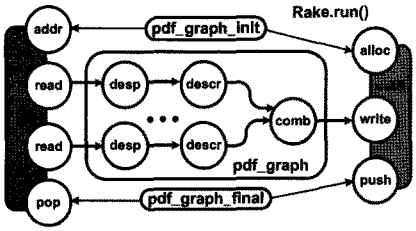
2.1장에서 언급되었듯이, 우리는 세 단계의 PDF 수행 모델을 제시하였다. 이 세 단계는 각각 세 개의 PDF 그래프 함수로 표현된다. pdf_graph_init는 PDF 수행의 초기화 단계를 나타내고, pdf_graph는 PDF 계산 단계를, pdf_graph_final은 PDF수행의 최종화 단계를 나타낸다. 하나의 PDF 그래프에 대해서, 이 세 가지 함수는 동일한 이름과 인자들을 가져야 한다. 그림 9은SPEX PDF 그래프로 구현된 W-CDMA의 rake 수신기를 보여준다. 세 가지 PDF 함수들은 각각 라인 15, 29, 및 39에 나와 있다. 이 예에서 Rake 수신기는 despreader, descrambler와 combiner 디지털신호처리 알고리즘으로 구성되어 있다. Despreader와 descrambler의 쌍은 rake finger라고 불리운다. Rake 수신기는 다음과 같은 런타임 재구성 요소를 지니고 있다. 첫째로, rake finger의 개수가 런타임시에 변할 수 있다. 둘째로, 한번에 스트리밍되는 데이터의 개수가 변할 수 있다. 셋째로, 각 rake finger의 스트리밍 상에서 읽는 주소가

변할 수 있다. Rake finger의 개수나 스트리밍의 크기는 각각 fingers와 stream_size라는 매개변수로 표현되었다. 스트리밍 상에서 읽는 주소는 PDF 수행 초기화 단계에서 입력 스트림 버퍼를 초기화 해줌으로써 변경할 수 있다(그림 9 라인 36).

**PDF 초기화 및 최종화** PDF 초기화의 목적은 PDF 그래프를 수행하도록 준비시키는 것이다. 본격적인 수행 중에는 SDF 가 되어야 하므로, 초기화 단계에 모든 매개변수들에 상수를 할당하여야 한다. 스트리밍 데이터 전달 패턴을 결정하는 것 또한 초기화 단계에 일어난다. 그림 9의 rake 수신기 예에서 라인 29부터 38까지 초기화 단계가 나타나 있다. 라인 31과 32에서는 입력 spreading factor에 의해 stream_size가 결정된다. 라인 35와 36에서는 각각의 finger의 스트리밍 상의 읽는 주소가 할당된다. 스트리밍 출력 버퍼를 셋팅하기 위해서, 라인 37에서처럼 메모리 공간을 할당한다.

PDF 최종화 단계의 목적은 데이터플로우 계산으로부터의 결과를 PDF 그래프의 내부 상태에 반영하는 것이다. 이는 또한, 그림 9의 라인 41,42,43에 보여지듯이, 입력 및 출력 버퍼의 메모리를 관리하는 것을 포함한다. 입력 버퍼로부터 모든 rake finger로부터 데이터가 읽혀지고 난 후에는, 버퍼를 위해 할당되었던 메모리가 회수된다. 출력 버퍼는 이 버퍼를 읽는 다른 PDF 액터들에게 데이터를 제공하기 위해서 푸쉬 오퍼레이션을 수행한다.

그림 10은 PDF 수행의 동시성을 표현하기 위한 문법 구문을 보여준다. pdf{} 구조는 PDF 그래프를 설명하기 위해 사용된다. pdf_graph 함수내에서 pdf{} 구조는 PDF 그래프에 해당하는 부분을 지정한다. 모든 pdf{} 구조는 지속적인 스트리밍 수행을 나타내는 for-문을 반드시 하나 가져야 한다. pdf{} 구조내에서 for-문에 선행하는 부분과 후행하는 부분은 각각 함축적으로 표현된 PDF 초기화 단계와 최종화 단계이다. PDF 초기화 및 최종화 계는 그림 9(b)에서처럼 명시적으로 표현될 수 도 있고 그림 11(c)에서 보여지듯이 함축적으로 표현될 수 있다. 또한 pdf{} 구조는 그 안에 또다른 pdf{} 구조를 포함할 수 있다. ll_for 구조는




(a) Rake receiver diagram

```
01 class Rake: spex_kernel {
02  private:
03   channel<int> chan1[MAX_FINGERS];
04   channel<int> chan2[MAX_FINGERS];
05   channel<int> chan3[MAX_FINGERS];
06   channel<int> chan4;
07   Despreader<int> despreaders[MAX_FINGERS];
08   Descrambler<int> descramblers[MAX_FINGERS];
09   Combiner<int> combiner;
10   param<int> stream_size;
11   ...
12  public:
13   Rake() { stream_size.values(2, 1000, 2000); }
14   ...
15   pdf_graph(run)(Buffer inb, Buffer& outb,
16                  param<int> r, param<int> fingers) {
17     pdf {
18       for (int j = 0; j < stream_size; j++) {
19         ll_for(int i = 0; i < fingers; i++) {
20           inb.read(chan1[i]);
21           despreader[i].run(chan1[i], chan2[i]);
22           descrambler[i].run(chan2[i], chan3[i]);
23         }
24         combiner.run(chan3, chan4);
25         outb.write(chan4);
26       }
27     }
28   }
29   pdf_graph_init(run)(Buffer inb, Buffer& outb,
30                       param<int> r, param<int> fingers) {
31     if (r == 4)      stream_size = 1000;
32     else if (r == 8) stream_size = 2000;
33     inb.num_readers(fingers);
34     combiner.num_rake_fingers(fingers);
35     for (int i = 0; i < fingers; i++)
36       inb.reader_start_addr(i, peak[i]);
37     outb.write_alloc(250);
38   }
39   pdf_graph_final(run)(Buffer inb, Buffer& outb,
40                        param<int> r, param<int> fingers) {
41     if (r == 4)      inb.pop(1000);
42     else if (r == 8) inb.pop(2000);
43     outb.push(250);
44   }
45 };
```

(b) Rake receiver SPEX implementation

그림 9 PDF 그래프 함수를 이용한 rake 수신기 구현의 예

for문의 바디가 병렬적으로 수행될 수 있음을 나타낸다. 즉, for-문의 매 회가 서로 독립적임을 표현한다. 그림

| SPEX 문법 구조 | 설명 |
|---|---|
| pdf{<br>//PDF 초기화 코드<br>for(…) {…} //PDF 수행 코드<br>// PDF 최종화 코드<br>} | Pdf 구조체.<br>반드시 하나의 for-문을 지녀야 한다. |
| ll_for(; ;){<br>… //loop body<br>} | 병렬적인 수행을 나타내는 for-문. 각 iteration 은 병렬적으로 수행될 수 있다. 서로 다른 iteration 간에 데이터 의존이 있어선 안된다. |

그림 10 동시성을 나타내기 위한 SPEX 문법 구조

9의 라인 19부터 24까지에 예시되어 있다.

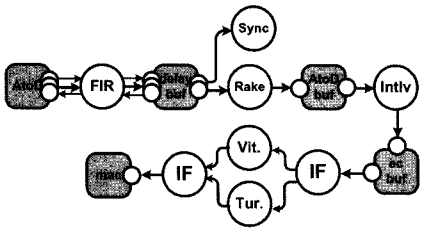
**SPEX PDF 그래프 언어의 한정** 런타임시에 SDF로 수행되도록 하기 위해서, 다음과 같은 한정이 적용된다. PDF 그래프 내에서 사용되는 모든 PDF 액터, 채널, 버퍼, 메모리 액터들은 PDF 그래프 클래스의 private 변수로 선언되어야 한다. C++에서와 동일한 객체의 배열을 선언할 수 있으나, 반드시 정적으로 선언하여야 한다. 매개변수들 또한 반드시 private 멤버로서 선언되어야 하며, PDF 함수의 지역 변수로 선언되어서는 안된다. 매개변수의 값은 반드시 `pdf_graph_init` 함수 안에서나, `pdf{}` 구조의 `for` 문 앞의 함축적인 초기화 단계에 상수로 정의되어야 한다. 또한 이러한 매개변수의 값은 `for` 문의 바디에서 재정의 되는 것이 허용되지 않는다.

**W-CDMA 수신기 구현의 예** 그림 11은 C언어 및 SPEX로 구현된 간단한 W-CDMA 수신기의 예를 보여준다. SPEX를 이용하면, C 언어만을 이용할 때보다 코드의 크기가 커진다. SPEX에서는 스트리밍 계산 자체에 26줄의 코드가 사용되었고, C 언어에서는 15줄만이 사용되었다. 그러나 C언어를 이용한 구현에서는 스트리밍 특성이 훼손되었다. 알고리즘이 순차적으로만 수행되어야 하므로 전체적인 스트림의 중간 결과를 모두 넘길 수 있도록 대용량의 버퍼가 할당되었다. C 코드를 다시 고쳐씀으로써 버퍼의 크기를 줄일 수는 있다. 그러나 최적의 버퍼 크기는 하드웨어 메모리 사이즈의 영향도 받기 때문에 프로그래머는 기계에 의존적인 코드를 작성할 수 밖에 없다. SPEX를 이용하면 스트리밍 형태가 프로그램 자체에 드러나므로, 컴파일러가 자동적으로 최적의 버퍼 크기를 결정할 수 있다. 프로그래머는 하드웨어에 관련한 지식까지 동원할 필요가 없다.

W-CDMA의 표준은 수신하는 데이터를 TTI(Transmission Time Interval) 블록으로 나누어 처리한다. 각 TTI 블록은 최대 5개의 W-CDMA 프래임을 가지고, 각 프래임은 15개의 슬롯으로 이루어져 있다. 데이터플로우의 재구성은 여러 개의 블록마다 일어난다. 각 PDF 그래프의 재구성은 자체의 초기화 단계 및 최종화 단계를 필요로 하기 때문에, 그림 11(c)에 나타난 것처럼 `pdf{}` 구조가 중첩되어 나타날 수 있다.

## 4. 기존 연구

데이터플로우 계산 모델 재구성 가능한 데이터플로우 계산 모델에 대한 다양한 연구가 있어 왔다. Finite State Machine(FSM)을 가지는 하이브리드 SDF[7]과 매개변수 SDF(PSDF)[8] 등이 그것이다. 전통적인 SDF에 비해 동적인 데이터플로우 모델로는 순환-정적




(a) W-CDMA reciever PDF diagram

```
01 void WCDMA_Receiver(int* AtoD, int* mac)
02 {
03   int delay_buf[slot_size];
04   int itlv_buf[slot_size];
05   int ec_buf[slot_size];
06
07   for (int i=0; i<15; i++) {
08     addr = 0;
09     for (int j=0; j<slot_size; j++) {
10       int data = AtoD[addr];
11       data = fir_run(data);
12       delay_buf[addr++] = data;
13     }
14     fingers = sync_run(delay_buf);
15     rake_run(delay_buf, itlv_buf, fingers);
16     interleaver_run(itlv_buf, ec_buf);
17     if (mode == voice)
18       viterbi_run(ec_buf, mac);
19     else if (mode == data)
20       turbo_run(ec_buf, mac);
21   }
22 }
```

Must allocate large buffers for the entire stream

Imperative computation descriptions

(b) W-CDMA reciever C implementation

```
01 class WCDMA_Receiver: spex_kernel {
02  private:
03   FIR<int> fir;
04   Rake rake;
05   // ... other DSP kernel declarations
06
07  public:
08   pdf_graph(run)(Buffer AtoD, Buffer& mac) {
09     pdf {
10       for (j = 0; j < 15; j++) {
11         pdf {
12           AtoD.read_addr(0);
13           delay_buf.push_alloc(slot_size);
14           chan1.src_bind(AtoD.reader);
15           chan2.dst_bind(delay_buf.writer);
16
17           for (i = 0; i < slot_size; i++) {
18             AtoD.read(chan1);
19             fir.run(chan1, chan2);
20             delay_buf.write(chan2);
21           }
22
23           AtoD.pop(slot_size);
24           delay_buf.push(slot_size);
25         }
26         sync.run(delay_buf, fingers);
27         rake.run(delay_buf, itlv_buf, rate, fingers);
28         Interleaver.run(itlv_buf, ec_buf);
29         if (mode == voice)
30           viterbi.run(ec_buf, mac);
31         else
32           turbo.run(ec_buf, mac);
33       }
34     }
35   }
36
37   pdf_graph_init(run)(Buffer AtoD, Buffer& mac) { ... }
38   pdf_graph_final(run)(Buffer AtoD, Buffer& mac) { ... }
39 };
```

Implicit PDF initialization

Implicit PDF finalization

Nested PDF graph description with implicit initialization and finalization

(c) W-CDMA reciever SPEX implementation

그림 11 W-CDMA 수신기 구현의 예: (b) C 구현, (c) SPEX 구현

데이터플로우 모델(cyclo-static dataflow)[6], 불린 데이터플로우 모델(BDF: Boolean Dataflow)[9], 그리고 synchronous piggybacked 데이터플로우(SPDF)[10] 등이 있다. CSDF는 순환적인 데이터플로우 속도를 지원한다. BDF는 조건에 따라 분기하는 액터를 지원한다. SPDF는 데이터플로우에 컨트롤플로우를 결합시킴으로써 데이터플로우의 재구성을 지원한다. FSM과 SDF의 하이브리드 모델은 FSM의 서로 다른 상태에 따라 서로 다른 종류의 데이터플로우 모델을 지원한다. SPEX의 PDF모델은 PSDF와 매우 비슷하다. PSDF와의 큰 차이점 중의 하나는 메모리 액터를 지원하여 데이터가 공유될 수 있도록 한다는 점이다. 다중-속도의 디지털신호처리 어플리케이션을 위해서 계층적인 데이터플로우 모델 또한 제안되었다[11].

**데이터플로우 언어** 디지털신호처리 시스템을 모델하기 위한 언어적 접근 또한 있어 왔다. 여러 가지 데이터플로우 모델을 기반으로 다양한 범위의 어플리케이션을 지원하기 위한 프레임워크들로는 Ptolemy project[8], DIF format[12], PeaCE design flow[13] 등이 있다. StreamIt처럼 특정한 아키텍처용 프로그래밍 언어로 디자인된 경우도 있다. StreamIt은 스트리밍 계산을 타일화된(tiled) 프로세서 아키텍처에 매핑하기 위하여 제안되었다[14].

StreamIt은 SDF에 기반한 스트리밍 언어이다. StreamIt에서는 SDF 액터 액터의 토큰 생성과 소모가 명시적으로 언어 상에 드러난다. 또한 액터들 간에 연결된 구성, 즉 데이터플로우 그래프가 명시적으로 드러난다. 그러나 StreamIt은 기존의 명령형 언어에 더하여 쓰여지기 위한 목적이 아니고, 스트리밍을 위한 새로운 언어라고 볼 수 있다. 따라서 새로운 언어에 익숙해지기 위한 프로그래머의 노력이 더 필요하다. 액터들 간의 데이터 전달 또한 선입선출법에 의해서만 표현되므로, 복잡한 데이터 전달 패턴을 표현하는 데에도 역시 선입선출법에 기반하여 재구성 해 주어야 한다. 또한, 최근에는 teleport 메세지를 통한 제한적인 동적인 데이터플로우 표현으로 확장되고 있으나, 기본적으로 StreamIt은 SDF에 기반한 정적인 데이터플로우 표현에 적합하다.

반면에, SPEX는 SDF보다는 동적인 데이터플로우를 표현할 수 있으면서도, 구현의 용이를 고려한 PSDF에 기반하고 있다. 초기 프로그래밍 노력을 줄이기 위해, 이미 디지털신호처리 시스템에 오랫동안 사용되어온 C/C++언어에 키워드를 더하는 방식으로 이루어 있으며, 이미 구현된 시스템에 사용된 복잡한 데이터 전달 방식을 그대로 구현할 수 있도록 메모리 액터 등을 보강하였다.

그 밖의 스트리밍 언어들 이 외에도 데이타플로우에 기반하지 않은 스트리밍 언어들도 있는데, Brook[15]과 Sequoia[16]가 그것이다. 이들은 모두 명령형 언어로서, 명시적인 스트리밍 배열 구조를 가진다. 또한 Sequoia는 어플리케이션의 메모리 계층구조가 프로그래머에게 드러나도록 디자인되었다.

## 5. 결 론

이 논문에서 우리는, 스트리밍 디지털신호처리 시스템의 계산 및 데이터 전달 속성을 지원하기 위한 SPEX의 언어적 특성을 설명하였다. SPEX의 의미구조는 매개변수 데이터플로우 모델에 기반한다. 우리는 이 매개변수 데이터플로우 모델을, 액터들 간에 메모리 공유가 가능하도록 수정하였다. 이를 통해, 데이터플로우 액터들의 조합으로 복잡한 데이터 전달 패턴을 표현하는 것이 가능해졌다. SPEX는 C++ 언어에 적용되었다. 이는 매개변수 데이터플로우를 표현하기 위한 키워드들과, 호스트 언어인 C++ 의 의미구조 한정으로 이루어져 있다.

## 참 고 문 헌

[1] B. Bhattacharya and S. S. Bhattacharyya. Parameterized Dataflow Modeling for DSP Systems. pages 2408-2421. IEEE Transactions on Signal Processing, Oct. 2001.

[2] E. Lee and D. Messerschmidt. Synchronous Data Flow. In Proc IEEE, 75, 1235-1245 1987.

[3] Y. Lin et al. SPEX: A Programming Language for Software Defined Radio. In Software Defined Radio Technical Conference and Product Exposition, 2006.

[4] Y. Lin, Y. Choi, S. Mahlke, T. Mudge, and C. Chakrabarti. A parameterized dataflow language extension for embedded streaming systems. In Proc. of the 2008 Systems, Architectures, Modeling, and Simulation (SAMOS), July 2008. pp.10-17.

[5] P. Murthy and E. A. Lee. Multidimensional Synchronous Dataflow. In IEEE Transactions on Signal Processing, August 2002.

[6] T. M. Parks, J. L. Pino, and E. A. Lee. A Comparison of Synchronous and Cyclo-Static Dataflow. In Asiolmar Conference on Signals, Systems and Computers, October 1995.

[7] L. Thiele, K. Strehl, D. Ziegenbein, R. Ernst, and J. Teich. FunState - An Internal Representation for Codesign. Proc. International Conference on Computer Aided Design, Nov. 1999.

[8] E. A. Lee. Overview of the Ptolemy Project. In Technical Memorandum No. UCB/ERL M03/25, University of California, Berkeley, July 2003.

[9] J. T. Buck and E. A. Lee. Scheduling Dynamic Dataflow Graphs With Bounded Memory Using the Token Flow Model. Proc. Int. Conf. Acoust.,

Speech, Signal Processing, April 1993.
[10] C. Park, J. Chung, and S. Ha. Extended Synchronous Dataflow for Efficient DSP System Prototyping. pages 295-322. Design Automation for Embedded Systems, Kluwer Academic Publishers, March 2002.
[11] N. Chandrachoodan and S. S. Bhattacharyya. The Hierarchical Timing Pair Model for Multirate DSP Applications. In IEEE Transactions on Signal Processing, volume 52, no. 5, May 2004.
[12] C. Hsu, I. Corretjer, M. Ko, W. Plishker, and S. S. Bhattacharyya. Dataflow interchange format: Language reference for DIF language version 1.0, users guide for DIF package version 1.0. In Technical Report UMIACS-TR-2007-32, Institute for Advanced Computer Studies, University of Maryland at College Park, June 2007.
[13] S. Ha, S. Kim, C. Lee, Y. Yi, S. Kwon, and Y. Joo. PeaCE: A Hardware-Software Codesign Environment for Multimedia Embedded Systems. In ACM Transactions on Design Automation of Electronic Systems, Aug. 2007.
[14] W. Thies, M. Karczmarek, and S. Amarasinghe. StreamIt: A Language for Streaming Applications. In Proc. of the 2002 International Conference on Compiler Construction, June 2002.
[15] I. Buck. Brook Language Specification. In http://merrimac.stanford.edu/brook, Oct. 2003.
[16] K. Fatahalian et al. Sequoia: Programming the Memory Hierarchy. In Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, Nov. 2006.

최 윤 서

2000년 한동대학교 전산전자 공학부 학사. 2002년 한국과학기술원 전산학과 석사. 2007년 한국과학기술원 전산학과 박사. 2008년~현재 미국 University of Michigan EECS 박사후 연구원. 관심분야는 멀티코어 컴파일러, 스트리밍 컴파일러 등

Yuan Lin

2002년 미국 Cornell University 학사 2008년 미국 University of Michigan 박사. 현재 미국 Cavium Networks 재직 중. 관심분야는 멀티코어 컴파일러, 저전력 멀티코어 아키텍처 디자인 등