Marco Formal para la Especificación de Implementaciones de la FFT sobre Transputers

Carlos Barra Peñaloza
Armada de Chile
Valparaíso, Chile
cbarra@inf.utfsm.cl

Horst von Brand
Universidad Técnica Federico Santa María
Valparaíso, Chile
vonbrand@inf.utfsm.cl

RESUMEN

Se presenta un marco de especificación formal, basado en TLA$^+$, para especificar topologías de redes de transputers que implementen el algoritmo de la Transformada Rápida de Fourier (FFT) En este marco se especifican el algoritmo y la arquitectura de los transputers, aislando los aspectos que definen una topología particular. Dichos aspectos son considerados en forma abstracta, lo que permite especificar una implementación dada mediante un módulo de TLA$^+$ independiente. Este trabajo se basa en [BAR96].

Introducción

La vasta investigación que se ha realizado sobre la computación paralela de la FFT incluye el desarrollo de redes de interconexión particularmente adecuadas para el intercambio de datos requerido por el algoritmo; mediciones de rendimiento de distintos sistemas paralelos (memoria compartida, memoria privada con red de interconexión, vectores, etc.); y el estudio del impacto del costo de comunicación sobre el rendimiento global, para diferentes estrategias de interconexión (ver referencias citadas en [MAV95], [FRA94], y [QUI93]). Actualmente, un aspecto importante de dicha investigación, es la explotación de los últimos avances en el diseño y fabricación de circuitos VLSI ("Very Large System Integration"), pero también existe un considerable interés en el uso de multicomputadores MIMD ("Multiple Instructions Multiple Data") donde la DFT puede ser integrada con otros algoritmos de procesamiento.

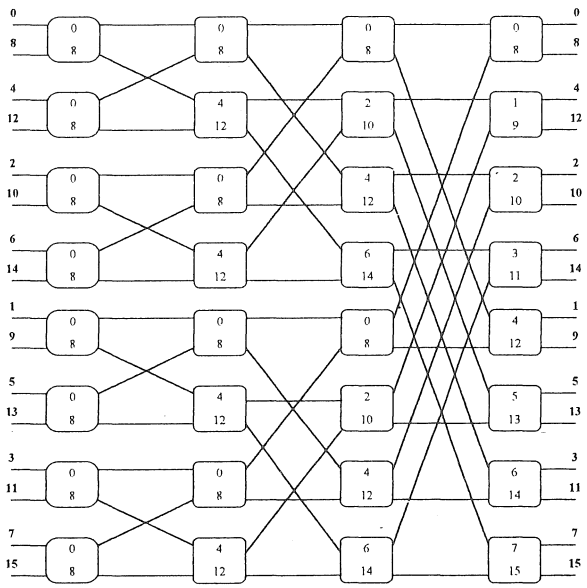
1. La Transformada Rápida de Fourier (FFT)

La transformada discreta de Fourier (DFT –"Discrete Fourier Transform") es una herramienta muy utilizada en el procesamiento de señal, en que el algoritmo comúnmente

usado para su computación es la FFT ([SDS75]). A pesar de la drástica reducción en el número de operaciones de punto flotante que se ha logrado mediante técnicas de transformación, la FFT continúa siendo una tarea intensiva en operaciones. Existen muchas aplicaciones donde los datos a ser transformados son producidos a velocidades muy altas (sistemas de tiempo real. Por otra parte, la FFT no sólo es un algoritmo intrínsecamente paralelo, si no que además su computación es completamente independiente de los datos.. La DFT transforma un vector de N componentes reales,, en un vector de N componentes complejos, mediante la multiplicación del primero por una matriz de N por N de las potencias de la N-ésima raíz primitiva de la unidad ([SBA83]). Un algoritmo secuencial, mediante el cual la FFT realiza el cálculo de la DFT, se basa en la *descomposición temporal* de los elementos del vector de entrada, lo que significa que éstos son reordenados de acuerdo a la *inversión de bits* (*bit reversal*) ([AHU74], [SDS75]), y en la estrategia de dividir y conquistar para la evaluación en los diferentes niveles de descomposición. Este algoritmo puede ser visualizado explícitamente mediante un diagrama de flujo de señal como el que se muestra en la Figura 1.1, donde se ha considerado el caso específico para $N = 16$, lo que implica 4 niveles de descomposición ($\log_2 N$).

Figura 1.1:

Diagrama de flujo de señal para N = 16



El diagrama de flujo de señal de la Figura 1.1 no permite incorporar formalmente aspectos importantes necesarios para un análisis de rendimiento, que son la arquitectura de la plataforma donde será implementado el algoritmo, la granularidad del paralelismo de la implementación y la topología de la arquitectura. Estos aspectos no son independientes entre sí y tampoco son únicos, por lo que se requiere de una especificación formal que permita introducir modificaciones y así poder realizar el análisis comparativo de diversas combinaciones de estos factores.

TLA^+ ([LLM95]) es un lenguaje formal basado en la lógica temporal de acciones (TLA –"Temporal Logic of Actions", [LLA94]) que permite especificar mediante componentes denominados módulos, que normalmente se utilizan para representar la descomposición de un sistema; por lo cual, la incorporación de las especificaciones de los componentes arquitectura, paralelismo y topología, se realizará mediante dicho lenguaje.

2. Especificación de una Arquitectura de Multicomputadores

Un transputer ([INM89]) es un componente VLSI con procesador, memoria y enlaces (*links*) de comunicaciones para conectarse directamente con otro, usando un protocolo tipo *rendezvous*. Se puede suponer que ([MAV95]) las comunicaciones a través de cada *link* ocurren concurrentemente con las de otros *links* y con la actividad de CPU ([INM89]).

3. Especificación de Aspectos Lógicos de la Arquitectura

El protocolo de comunicaciones se establece entre dos procesos, lo que plantea la necesidad de identificar los procesos tanto en forma individual, como en pares de procesos que se comunican. Esto lleva a considerar dos aspectos para la especificación de la arquitectura: es necesario implementar identificadores para los procesos (*Butterflies*), y especificar la restricción que impone el algoritmo para procesos que se comunican. Así, en la especificación del protocolo se tendrá, implícitamente, la especificación de los canales *virtuales*

de comunicación (que comunican procesos), que se diferencian de los canales reales en que estos últimos comunican procesadores. Dependiendo de la granularidad del paralelismo, cada procesador tendrá asignado un cierto número de procesos, cada uno de los cuales dispondrá de canales virtuales de comunicación implementados por canales reales, por lo que la topología involucra el mapeo de los procesos a los procesadores y la implementación de los canales virtuales.

Se visualizan tres módulos de especificación que se denominarán: *SetMap*, para especificar la identificación individual de procesos y procesadores y para definir la función de mapeo requerida por la especificación de la topología; *ProcessComm*, para especificar los pares de procesos que se comunican; y *Rendezvous*, para especificar el protocolo en base a los dos módulos anteriores. Nótese que en esta etapa no se pretende especificar la topología, sólo se especificarán las funciones necesarias para ello.

El protocolo para la comunicación entre un par de procesos debe contar con una variable que permita sincronización y una variable que represente el mensaje, las cuales deben ser del tipo correcto. La Figura 3.1 especifica el módulo *Types* que declara dicha correctitud. En la Figura 3.2 se presenta el módulo *SetMap*, que especifica cada procesador como un conjunto de procesos asignados mediante la función *Map* (que debe ser implementada como parte de la especificación de la topología). Así, cada proceso queda formalmente identificado por $\langle i, level \rangle$ y cada procesador por $\langle j, stage \rangle$, siendo *Map* una función que mapea un par identificador de proceso a un par identificador de procesador (el que ejecuta el proceso cuyo par identificador es mapeado).La Figura 3.3 muestra el módulo *ProcessComm*, que especifica un proceso como un par de variables $\langle ctrl, msg \rangle$, y que el conjunto *InterComm* está formado por los pares de procesos que se comunican en el algoritmo, cuyos pares identificadores están dados en base a las funciones *CommDown* y *CommUp* especificadas.

Figura 3.1:
Módulo de Especificación de Tipos

Module *Types*

definitions LegalValue ≡ COMPLEX
Control ≡ {"rtgd", "rtg", "rtr", "rtw"}

La Figura 3.4 muestra el módulo *Rendezvous* que especifica dos operaciones disponibles para los procesos (el ambiente de este componente): *Read* y *Write*; estas operaciones tienen dos parámetros: el canal virtual (*cp*) y el valor (*val*) que se almacena o se obtiene, respectivamente, de una variable buffer que contiene cada proceso (*msg*). Además, cada operación cambia el estado del proceso invocador (*ctrl* pasa "rtr" o "rtw") deteniendo las acciones de éste en espera de que se produzca el paso del mensaje (acción *MsgPass*). La propiedad de *fairness* para la acción de la acción *MsgPass*, asegura que ésta ocurrirá eventualmente. Así, el protocolo queda especificado por $Comm_{CH}$, y la arquitectura por *Arq*. Nótese que, de acuerdo a la arquitectura de los transputers ([INM89]), la especificación del protocolo no evita situaciones de bloqueo tales como cuando un proceso ejecuta la operación Read o Write sin que el proceso al cual va dirigido dicha operación, ejecute la operación Write o Read, respectivamente, hacia el primero. Por tanto, depende de la especificación del sistema asegurar la condición WF de la acción *MsgPass*.

Figura 3.2:
Módulo de Especificación de Identificación de procesos y procesadores

Module *SetMap*

import *Naturals*
parameters *I*, *LEVEL*, *J*, *STAGE* : constant
i, *level*, *j*, *stage* : variable

definitions $Element(M, N) \equiv \{\langle i, j\rangle : (0 \leq i \in \text{Nat} < M) \wedge (0 \leq j \in \text{Nat} < N)\}$
$Matrix(Set) \equiv \text{union}\{[Element(M, N) \rightarrow Set] : M, N \in \text{Nat}\}$
$Mapping \equiv [Element(I, LEVEL) \rightarrow Element(J, STAGE)]$
$t[j, stage] \equiv \{ p[i, level] : \wedge p \in \text{Matrix}(P)$
$\wedge \langle i, level\rangle \in \text{domain}(p)$
$\wedge t \in \text{Matrix}(T)$
$\wedge \langle j, stage\rangle \in \text{domain}(t)$
$\wedge Map \in Mapping$
$\wedge \langle j, stage\rangle = Map[i, level] \}$

Figura 3.3: *Módulo de especificación de proceso e intercomunicaciones*

Module *ProcessComm*

import *SetMap, Types*

definitions

$$P \equiv \left\{ \langle p[i, level].ctrl, p[i, level].msg \rangle : \begin{array}{l} \wedge\ msg \in \text{LegalValue} \\ \wedge\ ctrl \in \text{Control} \end{array} \right\}$$

$$CommDown[i, level] \equiv i - \left((i \bmod 2^{level+1}) - (i \bmod 2^{level}) \right)$$

$$CommUp[i, level] \equiv CommDown[i, level] + 2^{level}$$

$$InterComm \equiv \bigcup_{\text{domain}(p)} \left\{ \begin{array}{c} \{\langle p[i, level], p[CommUp[i, level], (level + 1)] \rangle\} \\ \cup \\ \{\langle p[i, level], p[CommDown[i, level], (level + 1)] \rangle\} \end{array} \right\}$$

Figura 3.4: *Módulo de especificación del protocolo rendezvous*

Module *Rendezvous*

import *SetMap, ProcessComm, Types*

parameter *val* : variable

assumptions $Asssump \equiv \wedge\ val \in \text{LegalValue}$
$\wedge\ \exists k \in \{0,1\} \wedge \exists \langle m,n \rangle \in \text{domain}(p) : cp[k] = p[m,n]$

environment definition

$$Init_{Ch}(cp) \equiv \forall k \in \{0,1\} : cp[k].ctrl = \text{"rtg"}$$

$$Read(cp, val) \equiv \exists k \in \{0,1\} : \begin{array}{l} \wedge\ cp[k].ctrl = \text{"rtg"} \\ \wedge\ cp[k].ctrl' = \text{"rtr"} \end{array}$$

$$Write(cp, val) \equiv \exists k \in \{0,1\} : \begin{array}{l} \wedge\ cp[k].ctrl = \text{"rtg"} \\ \wedge\ cp[k].ctrl' = \text{"rtw"} \\ \wedge\ cp[k].msg' = val \end{array}$$

$$Comm_E(cp, val) \equiv Read(cp, val) \vee Write(cp, val)$$

component definition

$$MsgPass(cp, val) \equiv \exists k \in \{0,1\} : \begin{array}{l} \wedge\ cp[k].ctrl = \text{"rtr"} \\ \wedge\ cp[1-k].ctrl = \text{"rtw"} \\ \wedge\ val' = cp[1-k].msg \\ \wedge\ cp[0].ctrl = \text{"rtg"} \\ \wedge\ cp[1].ctrl = \text{"rtg"} \\ \wedge\ \forall k : cp[k].msg' = \varnothing \end{array}$$

protocol definition

$$Comm_{Ch}(cp, val) \equiv \begin{array}{l} \wedge\ Init_{Ch}(cp, val) \\ \wedge\ \Box \left[\begin{array}{l} \vee\ Comm_E(cp, val) \\ \vee\ MsgPass(cp, val) \end{array} \right]_{\langle cp, val \rangle} \\ \wedge\ WF_{\langle cp, val \rangle}(MsgPass(cp, val)) \end{array}$$

$$Arq \equiv \bigwedge_{cp,\ val} Comm_{Ch}(cp, val)$$

4. Implementación del Sistema AA (Algoritmo-Arquitectura)

Se considerará un módulo *StateFunctions* para especificar las funciones de estado de la primera etapa y un módulo *FFT-Actions* para especificar las acciones del comportamiento de los procesos. Por otra parte, se especificará la integración de la arquitectura, pero, en realidad, la arquitectura se incorpora a la implementación a través de una topología, es decir, el protocolo de comunicaciones se manifiesta a través de la interacción de procesos, la que se produce sobre una red de conexiones físicas (topología de conexión –canales reales–) entre los procesadores a los cuales se les han asignado dichos procesos (topología de mapeo –asignación de procesos–), todo lo cual es lo que se considera como topología. Como no se pretende especificar una topología particular, se considerará una "topología virtual" sobre la cual los procesos se comunican a través de canales virtuales, bajo el protocolo de la arquitectura ya especificada. Entonces, un módulo *Virtual-Topology* contendrá la especificación de las acciones virtuales que esta topología virtual impone al comportamiento del sistema, y un módulo *SpecFFT-Virtual* contendrá la especificación del sistema algoritmo-arquitectura (AA).

La Figura 4.1. muestra el módulo *StateFunctions*, que especifica las funciones de estado necesarias para las acciones del algoritmo y para la identificación de procesos. *DatIn.Map* y *DatOut.Map* son funciones cuyos resultados son un conjunto de datos que se almacenan en un arreglo; *w.Map* es la función cuyos resultados son un conjunto de datos almacenados en una estructura tipo arreglo o tabla de valores; y *BitReversal* y *PowRoot* tienen la interpretación definida en el módulo *StateFunctions*. *F* es un conjunto de funciones que permite formalizar la notación tipo X [*i, level*] utilizada en la especificación de las acciones del algoritmo, como se muestra en la especificación del módulo *FFT-Actions* en la Figura 4.2. Los resultados se envían de un nivel a otro a través de un paso de mensaje realizado bajo el protocolo especificado por la arquitectura; lo cual se especifica en el módulo *Virtual-Topology* de la Figura 4.3. Esta topología virtual especifica que los procesos se comunican mediante *Rendezvous* para el paso de resultados, de modo que la combinación de los módulos *Virtual-Topology* y *FFT-*

Actions, representa todas las acciones del sistema, el cual se especifica en módulo *SpecFFT-Virtual* de la Figura 4.4. Esta especificación implementa al algoritmo considerando las restricciones que impone la arquitectura, por lo que queda especificado formalmente el sistema AA, representado por la fórmula *FFT-Virtual*; además, la operación *Butterfly* especifica la granularidad básica que se ha considerado y que deberá ser mantenida en la especificación de la topología de mapeo (para otras granularidades, habrá que modificar esta especificación y las siguientes). Así, en este nivel de abstracción, se tiene la especificación del sistema incluyendo algoritmo, arquitectura y topología virtual; esta última, considera algún tipo de mapeo procesos-procesadores y algún tipo de red de conexión entre procesadores, que la implementan de modo que *FFT-Virtual* es válida.

Figura 4.1: *Módulo de especificación de funciones de estado*

Module *StateFunctions*

import *Naturals*

parameters N : constant

F, f, m, n : variables

assumption $ParamAssump \equiv M, m, n \in \text{Nat}$

definitions $BitReversal[m] \equiv \sum_{k=0}^{(log_2 N)-1} \left(2^m\right) \cdot \frac{m \bmod 2^{((log_2 N)-k)} - m \bmod 2^{((log_2 N)-(k+1))}}{2^{((log_2 N)-(k+1))}}$

include *SetMap* **as** *DatIn* **with** $I \leftarrow N, LEVEL \leftarrow 1$:

$\forall m \in \text{domain}\left(DatIn.Map\right) : DatIn.Map[m] \in \text{LegalValue}$

include *SetMap* **as** *DatOut* **with** $I \leftarrow N, LEVEL \leftarrow 1$:

$\forall m \in \text{domain}\left(DatOut.Map\right) : DatOut.Map[m] \in \text{LegalValue}$

include *SetMap* **as** w **with** $I \leftarrow N/2 - 1, LEVEL \leftarrow 1$:

$\forall m \in \text{domain}\left(w.Map\right) : w.Map[m] = e^{-(\sqrt{-1})2\pi n/N}$

$PowRoot[m,n] \equiv w.Map\left[N \cdot 2^{-(m+1)} \cdot m \bmod 2^n\right]$

$F[m,n] \equiv \left[\langle m,n\rangle \rightarrow p[m,n].f\right]$

Figura 4.2: *Módulo de especificación de acciones*

Module *FFT–Actions*

import *Types, StateFunctions, ProcessComm, Rendezvous*

parameters x, y : variables

$result$: boolcan

assumption $ParamAssump \equiv \wedge\, x, y \in \text{LegalValue}$
$\wedge\, cp \in InterComm : cp[0] = p[i, level]$
$\wedge\, Ctrl \in F : Ctrl[i, level] = p[i, level].ctrl$
$\wedge\, Result \in F : Result[i, level] = p[i, level].result$
$\wedge\, X \in F : X[i, level] = p[i, level].x$
$\wedge\, Y \in F : Y[i, level] = p[i, level].y$

definitions

$$Init(i, level) \equiv \begin{pmatrix} \wedge\, Ctrl[i, level] = \text{"rtg"} \wedge Ctrl[i, 0] = \text{"rtgd"} \\ \wedge\, Result[i, level] = \text{False} \\ \wedge\, \text{if}\,(CommDown[i, level - 1] = i)\ \text{then}\ (r = 1)\ \text{else}\ (r = -1) \end{pmatrix}$$

$$GetDat(i, level) \equiv \begin{pmatrix} \wedge\, Ctrl[i, level] = \text{"rtgd"} \wedge Result[i, level] = \text{False} \\ \wedge\, X'[i, level] = DatIn.Map[BitReversal[i]] \\ \wedge\, Y'[i, level] = DatIn.Map[BitReversal[i] + N/2] \\ \wedge\, Ctrl'[i, level] = \text{"rtg"} \end{pmatrix}$$

$$CalcFFT(i, level) \equiv \begin{pmatrix} \wedge\, Ctrl[i, level] = \text{"rtg"} \wedge Result[i, level] = \text{False} \\ \wedge\, X'[i, level] = X[i, level] + Y[i, level] \cdot PowRoot[i, level] \\ \wedge\, Y'[i, level] = X[i, level] - Y[i, level] \cdot PowRoot[i, level] \\ \wedge\, \text{if}\,(level = 0) \\ \quad \text{then}\ Ctrl'[i, level] = \text{"rtgd"} \\ \wedge\, Result'[i, level] = \text{True} \end{pmatrix}$$

$$PutDat(i, level) \equiv \begin{pmatrix} \wedge\, (Result[i, level] = \text{True} \wedge level = \log_2 N) \\ \wedge\, DatOut.Map'[BitReversal[i]] = X'[i, level] \\ \wedge\, DatOut.Map'[BitReversal[i] + N/2] = Y'[i, level] \\ \wedge\, Result'[i, level] = \text{False} \end{pmatrix}$$

Figura 4.3: *Módulo de especificación de la topología virtual*

Module *Virtual - Topology*

import *FFT - Actions, ProcessComm*

definitions

$$ReceiveResult(i, level) \equiv \left(\begin{array}{l} \\ \wedge Ctrl[i, level] = \text{"rtg"} \\ \wedge Read(cp, X[i, level]) : \\ \quad cp[1] = p[CommDown[i, level-1], level-1] \\ \wedge Read(cp, Y[i, level]) : \\ \quad cp[1] = p[CommUp[i, level-1], level-1] \end{array} \right)$$

$$SendResult(i, level) \equiv \left(\begin{array}{l} \wedge Result[i, level] = \text{True} \\ \wedge Write(cp, X[i, level]) : \\ \quad cp[1] = p[CommDown[i, level], level+1] \\ \wedge Write(cp, Y[i, level]) : \\ \quad cp[1] = p[CommUp[i, level], level+1] \\ \wedge \text{if}(level \neq log_2 N) \\ \quad \text{then } Result'[i, level] = \text{False} \end{array} \right)$$

Figura 4.4: *Módulo de especificación de sistema AA*

Module *SpecFFT–Virtual*

import *FFT–Actions, Virtual-Topology*

definitions

$$Butterfly(i, level) \equiv \vee GetDat(i, level) \vee CalcFFT(i, level) \vee ReceiveResult(i, level) \vee PutDat(i, level) \vee SendResult(i, level)$$

$$hvar[i, level] \equiv \langle Ctrl(i, level), Msg(i, level), cp, Result(i, level) \rangle$$

$$vvar[i, level] \equiv \langle X(i, level), Y(i, level) \rangle$$

$$var[i, level] \equiv \langle hvar[i, level], vvar[i, level] \rangle$$

$$NodeFFT(i, level) \equiv Init(i, level) \wedge \Box[Butterfly(i, level)]_{var[i, level]}$$

$$FFT–Virtual \equiv \bigwedge_{i, level} (\exists hvar[i, level] : NodeFFT(i, level))$$

5. Especificación de Aspectos Físicos

Para especificar la red de conexiones físicas entre los transputers y el sincronismo de comunicaciones de una topología particular, se requiere la especificación de la figura 5.1, donde *Wire* es una función que permite una sola conexión abstracta de un *link* con otro de otro transputer; nótese que se incluye la posibilidad de que un transputer se comunique consigo mismo (el caso de dos procesos en un mismo procesador que se comunican). Un canal real de comunicación es aquel que existe entre dos procesos que se comunican, cuando ambos están asignados a procesadores (transputers) entre los que se cumple la función *Wire*. Así, la red de conexión entre transputers queda especificada como el conjunto *Channel* de pares de procesos *cp* que cumplen con la restricción anterior.

Figura 5.1: *Módulo de especificación de conectividad*

Module *Connectivity*

import *SetMap, ProcessComm*

parameters $j^*, stage^*, j^{**}, stage^{**}, cp$: variable

assumption *Assump* $\equiv \wedge\ cp \in InterComm$
$\wedge\ \langle j^*, stage^* \rangle, \langle j^{**}, stage^{**} \rangle \in \text{domain}\ (t)$
$\wedge\ \text{LINK} \equiv \{0, 1, 2, 3\}$
$\wedge\ link \in \text{LINK}$

definitions $TNet \equiv [\langle \text{domain}\ (t) \times LINK \rangle \rightarrow \langle \text{domain}\ (t) \times LINK \rangle]$

$Wire\ [j, stage, link] \equiv \langle j^*, stage^*, link^* \rangle$
$: \wedge\ Wire \in TNet$
$\wedge\ Wire\ [j^*, stage^*, link^*] = \langle j, stage, link \rangle$
$\wedge\ Wire\ [j, stage, link] = Wire\ [j^{**}, stage^{**}, link^{**}]$
$\Rightarrow \langle j, stage, link \rangle = \langle j^*, stage^*, link^* \rangle$

export

$$Connection\ [j, stage] \equiv \left\{ \begin{array}{l} \langle j^*, stage^* \rangle : \\ \left(\begin{array}{l} \exists\ link, link^* : \\ \langle j^*, stage^*, link^* \rangle = Wire\ [j, stage, link] \end{array} \right) \end{array} \right\}$$

$$Channel \equiv \left\{ \begin{array}{l} cp : \\ \left(\begin{array}{l} \exists\ k \in \{0,1\}, \langle j^*, stage^* \rangle \in Connection\ [j, stage] : \\ \left(\wedge\ cp\ [k] \in t\ [j, stage] \wedge\ cp\ [1-k] \in t\ [j^*, stage^*] \right) \end{array} \right) \end{array} \right\}$$

Los aspectos referidos (conexión y sincronismo de comunicaciones), se pueden resolver mediante la especificación del conjunto *Connection* y la especificación adecuada de las acciones *ReceiveResult* y *SendResult*, de modo que las operaciones *Read* y *Write* se ejecuten sobre *cp's* (canales virtuales definidos por pares de procesos que se comunican) del conjunto *Channel*. En una próxima etapa de especificación, se deberán especificar *Map*, *Connectivity*, *ReceiveResult* y *SendResult*, de acuerdo a la topología que se requiera; esto significa especificar cómo se asignan los procesos a los transputers, cómo estos transputers serán físicamente conectados y cómo los procesos se comunicarán a través de canales establecidos por dichas conexiones. Si se considera que estas especificaciones estarán contenidas en un módulo, entonces es posible hacer una especificación general del sistema AAT, importando dicho módulo.

6. Especificación del Sistema AAT (Algoritmo-Arquitectura-Topología)

Dado que se cuenta con la especificación de las acciones del sistema (módulo *FFT-Actions*) y la especificación de la topología (cualquiera sea), entonces, al igual que se especificó el sistema AA, se especifica el sistema AAT como se muestra en la Figura 6.1.

Figura 6.1: *Módulo de especificación del sistema AAT*

Module *SpecFFT*

import $FFT-Actions, Any-Topology$

definitions

$$Butterfly\ (i, level) \equiv \vee\ GetDat\ (i, level) \vee CalcFFT\ (i, level) \vee Any.ReceiveResult\ (i, level) \vee PutDat\ (i, level) \vee Any.SendResult\ (i, level)$$

$$hvar\ [i, level] \equiv \langle Ctrl\ (i, level), Msg\ (i, level), cp, Result\ (i, level) \rangle$$

$$vvar\ [i, level] \equiv \langle X\ (i, level), Y\ (i, level) \rangle$$

$$var\ [i, level] \equiv \langle hvar\ [i, level], vvar\ [i, level] \rangle$$

$$NodeFFT\ (i, level) \equiv Init\ (i, level) \wedge \Box\ [Butterfly\ (i, level)]_{var\ [i, level]}$$

$$FFT \equiv \bigwedge_{i, level} (\exists\ hvar\ [i, level] : NodeFFT\ (i, level))$$

La especificación del sistema AAT queda dada por la fórmula *FFT* d módulo *SpecFFT*, donde sólo basta reemplazar el nombre del módulo importado *Any-Topolog* por aquel del módulo de especificación de alguna topología particular. Por lo tanto, *FF* implementa a Φ si la especificación de la topología es válida, lo cual puede ser demostrad probando la validez de la fórmula $FFT \Rightarrow \Phi$, mediante los axiomas y reglas de demostración d TLA ([LLA94]), lo que se deja para un trabajo futuro por no corresponder al objetivo de este además, del mismo modo, se pueden probar las fórmulas *FFT-Virtual* $\Rightarrow \Phi$ y $FFT \Rightarrow$ *FFT-Virtual*. Así, sólo resta hacer una especificación válida de la(s) topología(s) en base a lo planteado en la sección anterior, para obtener la(s) especificación(es) completa(s) de el(los) sistema(s) AAT('s), para realizar un estudio de rendimiento.

Bibliografía

[AHU74] Aho A. V., J. E. Hopcroft, J. D. Ullman, "The Design and Analysis of Computer Algorithms", Addison-Wesley, Reading Mas., 1974.

[BAR96] Carlos Barra P,"Métodos formales en el Análisis y Diseño de Algoritmos Paralelos sobre Arquitecturas de Multicomputadores: un caso práctico ", Tesis de Magister en Ingeniería Informática, Universidad Técnica Federico Santa María, Valparaíso, Chile, 1996.

[FRA94] Paraskevi Fragopoulou, Selim G. Akl, "A Parallel Algorithm for Computing Fourier Transforms on the Star Graph", IEEE Transaction on Software Engineering, vol. 5, N° 5, May 1994.

[INM89] Inmos Ltd., "The Transputer Databook", Consolidated Printers, Berkeley, USA, 1989.

[LLA94] Leslie Lamport, "The Temporal Logic of Actions", ACM Transactions on Programming Languages and Systems, May 1994.

[LLM95] Leslie Lamport, "TLA$^+$", http://www.research.digital.com/SRC/personal/ Leslie_Lamport /tla/tla.html, 23 Mar 1995.

[MAV95] Antonio Mazzeo and Umberto Villano, "Parallel 1D-FFT Computation on Constant-valence Multicomputers", Software-Practice and Experience, vol. 25, June 1995.

[QUI93] Michael Quin, "Parallel Computing", McGraw-Hill Book Company, 1993.

[SDS75] Samuel D. Stearns, "Digital Signal Analysis", Hyden Books Inc., 1975.

[SBA83] Sara Baase, "Computer Algorithms: Introduction to Design and Analysis", Addison-Wesley, Reading Mas., 1983.