

**UNIVERSIDAD NACIONAL DE CORDOBA**  
**Facultad de Ciencias Exactas, Físicas y Naturales**

Laboratorio de Arquitectura de Computadoras



**Simulación de procesadores multicore para sincronizar  
múltiples procesos utilizando redes de Petri**

**Proyecto Integrador de Ingeniería en Computación**

**AUTORES**

Baldoni, Federico E. - Romano, Matías H.

**DIRECTOR**

Ing. Micolini, Orlando.

# ÍNDICE

---

Índice .....	i
Índice de imágenes.....	v
Resumen .....	vii
Motivación .....	ix
1    Introducción.....	1
2    Objetivos .....	3
2.1 Restricciones.....	3
3    Marco Teórico.....	5
3.1 Simulación y simuladores.....	5
3.1.1 SESC (5).....	7
3.2 Modelos.....	8
3.2.1 Autómatas .....	8
3.2.2 Redes de PETRI (7).....	10
3.2.3 PN vs. Autómata (6) .....	18
3.2.4 Seleccionando un modelo .....	18
3.3 Teoría de Sincronización.....	19
3.3.1 Exclusión Mutua .....	19
3.3.2 Condición de Sincronización .....	19
3.3.3 Solución a los tipos de Sincronización.....	19
3.3.4 Uso de redes de Petri en la construcción de monitores .....	22
3.4 Sincronización y Redes de Petri.....	24
3.4.1 Exclusión mutua entre secciones Críticas.....	24
3.4.2 Sincronización de tareas .....	24
4    Estudio Preliminar .....	27
4.1 Análisis de la Sincronización entre Hilos .....	30
4.1.1 Relación entre hilos sincronizados y sin sincronizar .....	31
4.1.2 Relación entre la cantidad de escrituras por sincronización respecto de hilos no sincronizados .....	32
4.2 Análisis de las alternativas de sincronización por hardware .....	34

5	Hipótesis .....	37
5.1	Resultados Esperados .....	37
6	Desarrollo e Implementación.....	39
6.1	Introducción a SESC.....	39
6.2	Algoritmo de Petri.....	40
6.3	Algoritmo de Petri Implementado .....	41
6.3.1	Implementación del Mecanismo en el Simulador.....	42
6.4	Simulaciones Y Mediciones .....	47
6.4.1	Escritor / Escritor .....	47
6.4.2	Productor / Consumidor.....	51
6.4.3	Algoritmo de Simulación de una Planta de embalaje (15).....	54
6.4.4	Algoritmo de control de un mecanismo de velocidad crucero con detector de distancia a objetos para automóviles.....	59
7	Conclusión.....	65
7.1	Comentarios Adicionales .....	65
8	Trabajos Futuros .....	67
9	Anexo I.....	69
9.1.1	¿Qué es un superescalar fuera de orden (out-of-order) pipeline? .....	69
9.1.2	¿Cómo modela esto SESC? .....	70
9.1.3	Ánálisis detallado del simulador.....	70
10	Anexo II.....	77
10.1	Instalación del Simulador SESC .....	77
10.1.1	Instalar gcc 3.4.....	77
10.1.2	Instalar Bison 2.3.....	79
10.1.3	Generar SESC Utils .....	79
10.1.4	Generar código fuente SESC .....	81
10.1.5	Pasos especiales para Ubuntu 10.10 de 64 bits .....	84
10.2	Compilar programa para simular con SESC .....	85
10.2.1	Programas multi-thread .....	86
10.3	Incorporar el simulador a la IDE KDevelop .....	86
10.3.1	Instalación de IDE KDevelop3 .....	86
10.3.2	Como Integrar el simulador SESC en KDevelop3 (18) .....	87
10.4	Mapeo de instrucciones MIPS-MINT .....	90
11	Anexo III.....	93
11.1	Archivo de configuración Original.....	93
11.2	Archivo de configuración Modificado .....	98
12	Anexo IV .....	103
12.1	Interpretación de los resultados de una simulación.....	103
12.2	Códigos y Mediciones.....	105
12.2.1	SINCRONIZACION.....	105
12.2.2	ESCRITOR/ESCRITOR.....	107
12.2.3	PRODUCTOR/CONSUMIDOR .....	119
12.2.4	CONTROL PLANTA DE EMBALAJE.....	132

---

12.2.5 CONTROL DE CRUCERO.....	140
13 Acrónimos .....	151
14 Bibliografía.....	153

DVD Adjunto: Código fuente SESC original y modificado, SESCUtils, gcc 3.4, bison 2.3.

Papers, a los que se puede acceder digitalmente, referenciados en la bibliografía.



# ÍNDICE DE IMÁGENES

---

FIGURA 1. SECUENCIA DE PASOS PARA SIMULAR EN SESC .....	8
FIGURA 2. AUTOMATA DE MÁQUINA EXPENDEDORA .....	9
FIGURA 3. EXTENSIONES A LOS AUTÓMATAS .....	9
FIGURA 4. ELEMENTOS DE UNA RED DE PETRI.....	10
FIGURA 5. RED DE PETRI DE LOS FILÓSOFOS CHINOS .....	14
FIGURA 6. CRONOLOGÍA DE LAS FAMILIAS DE PN.....	16
FIGURA 7. DOMINIOS DE LAS SEMÁNTICAS DE LOS MODELOS BASADOS EN PN .....	17
FIGURA 8. PN DE UN PRODUCTOR CONSUMIDOR .....	22
FIGURA 9. RED DE PETRI DE UNA EXCLUSIÓN MUTUA .....	24
FIGURA 10. REDES DE PETRI DE PRODUCTOR/CONSUMIDOR CON BUFFER LIMITADO A “N” UNIDADES.	25
FIGURA 11. ALGORITMO ESCRITOR/ESCRITOR (MÁXIMA SINCRONIZACIÓN VS. SIN SINCRONIZACIÓN) ..	32
FIGURA 12. RELACIÓN PORCENTUAL SINCRONIZAR VS. NO SINCRONIZAR .....	33
FIGURA 13. DIAGRAMA DE COMPONENTES SIMPLIFICADO DE SESC.....	40
FIGURA 14. MEMORIA CACHE A IMPLEMENTAR.....	42
FIGURA 15. DIAGRAMA DE FLUJO DEL FUNCIONAMIENTO DEL ALGORITMO DE PETRI IMPLEMENTADO.	44
FIGURA 16. DIAGRAMA DE FLUJO BÁSICO DE LA EJECUCIÓN DE UN PROGRAMA. ....	45
FIGURA 17. EJECUCIÓN DE DISPAROS POR 2 PROCESOS P1 Y P2. EL PROCESO P1 EFECTÚA EL DISPARO D1, CUANDO EL PROCESO P2 ESTÁ EJECUTANDO EL DISPARO D3, EL PROCESO P1 SE ENCUENTRA EN NS (NO SHOT) YA QUE LOS SUBPROCESOS A Y B NO SE PUEDEN DAR DE FORMA SIMULTÁNEA, DEBEN ESTAR SINCRONIZADOS. ....	46
FIGURA 18. RED DE PETRI QUE REPRESENTA LA ESCRITURA DE UNA VARIABLE COMPARTIDA. SIENDO “A” Y “B” LA PARTE DE LOS PROCESOS QUE ACCEDEN A LA VARIABLE COMPARTIDA. ....	47
FIGURA 19. COMPARATIVA ENTRE SIMULADOR ORIGINAL Y MODIFICADO RESPECTO DEL TIEMPO DE EJECUCIÓN .....	50
FIGURA 20. DIFERENCIA PORCENTUAL ENTRE SIMULADORES RESPECTO DE LAS INSTRUCCIONES EJECUTADAS .....	50
FIGURA 21. BUFFER LIMITADO PARA PROBLEMA PRODUCTOR/CONSUMIDOR.....	51
FIGURA 22. RED DE PETRI DEL MODELO PRODUCTOR/CONSUMIDOR .....	51
FIGURA 23. COMPARATIVA ENTRE SIMULADOR ORIGINAL Y MODIFICADO RESPECTO DEL TIEMPO DE EJECUCIÓN .....	53
FIGURA 24. DIFERENCIA PORCENTUAL ENTRE SIMULADORES RESPECTO DE LAS INSTRUCCIONES EJECUTADAS .....	54

FIGURA 25. REPRESENTACIÓN DEL MECANISMO DE EMBALAJE .....	55
FIGURA 26. RED DE PETRI DE LA PLANTA DE EMBALAJE .....	56
FIGURA 27. COMPARATIVA ENTRE SIMULADOR ORIGINAL Y MODIFICADO RESPECTO DEL TIEMPO DE EJECUCIÓN .....	58
FIGURA 28. DIFERENCIA PORCENTUAL ENTRE SIMULADORES RESPECTO DE LAS INSTRUCCIONES EJECUTADAS .....	59
FIGURA 29. INTERFAZ DE USUARIO DEL MECANISMO DE CONTROL DE VEL. CRUCERO .....	59
FIGURA 30. RED DE PETRI DEL SISTEMA.....	60
FIGURA 31. COMPARATIVA ENTRE SIMULADOR ORIGINAL Y MODIFICADO RESPECTO DEL TIEMPO DE EJECUCIÓN .....	63
FIGURA 32. DIFERENCIA PORCENTUAL ENTRE SIMULADORES RESPECTO DE LAS INSTRUCCIONES EJECUTADAS .....	63
FIGURA 33. RELACIÓN ENTRE LAS CLASES DEL PROCESADOR.....	71
FIGURA 34. JERARQUÍA DE RECURSOS.....	74
FIGURA 35. RESULTADO DE UNA SIMULACIÓN.....	103

## **RESUMEN**

---

Este trabajo posee la particularidad de innovar en el uso de las redes de Petri no solo como modelo de sistemas, sino como lenguaje de ejecución de algoritmos concurrentes. En él, se desarrolla un mecanismo para mejorar la sincronización entre procesos (hilos) a nivel de hardware, haciendo uso del formalismo de las redes de Petri (procesador de Petri) mediante el empleo de un simulador de procesadores, siendo este simulador el que implementa dicho mecanismo. Logrando así desarrollar una nueva arquitectura de procesadores.

El módulo desarrollado permite obtener mejoras tanto en tiempo como en cantidad de instrucciones ejecutadas para algoritmos concurrentes que requieren ser sincronizados, cuanto mayor es el uso de primitivas de sincronización en un algoritmo mayor es el beneficio del desarrollo en los puntos mencionados anteriormente. Las mejoras en tiempos de ejecución e instrucciones ejecutadas por el procesador simulado con el módulo de Petri tienen una disminución de entre un 25% y 50% respecto del procesador original.



## MOTIVACIÓN

---

La motivación del trabajo es articular una combinación de conceptos vistos en diferentes asignaturas de la carrera, con el fin de llevar a cabo la investigación, desarrollo e implementación de un mecanismo capaz de mejorar el rendimiento de los procesadores con múltiples cores de la actualidad, creando una nueva arquitectura de procesadores.



# 1 INTRODUCCIÓN

---

Los problemas de exclusión mutua y sincronización representan un gran aumento en los tiempos de ejecución de programas en sistemas computacionales (1). Existen diversos métodos para implementar la sincronización al acceso de recursos compartidos. Los más conocidos y desarrollados son Pipes, Semáforos y Monitores. Estas técnicas son implementadas por software, mediante intervención del sistema operativo para hacer la asignación de recursos, ocasionando así el aumento de los tiempos de ejecución. (2)

La herramienta utilizada para el modelado y resolución de problemas de concurrencia es el formalismo de Redes de Petri, pudiendo con esta modelar y resolver las técnicas mencionadas en el párrafo anterior. (3)

En este trabajo se propone mediante el uso de un simulador de procesadores superescalares, desarrollar e implementar una arquitectura de hardware la cual permita disminuir los tiempos de ejecución y la cantidad de instrucciones ejecutadas de los problemas de concurrencia y exclusión mutua sobre los recursos compartidos entre procesos y/o procesadores, utilizando Redes de Petri. Obteniendo de esa forma una nueva arquitectura de procesadores la cual ejecuta de manera más rápida aquellos programas que emplean sincronización.

El trabajo se encuentra dividido en las siguientes secciones: Objetivos, Marco Teórico, Estudio Preliminar, Hipótesis, Desarrollo e Implementación, Simulaciones y Mediciones, Anexos I, II, III y IV.

En los Objetivos se plantean estos junto a una serie de restricciones a tener en cuenta en la implementación.

Dentro del Marco Teórico se van a desarrollar conceptos teóricos necesarios para comprender el desarrollo del trabajo. Entre los conceptos más importantes se encuentran: porque simular la implementación y sobre qué simulador realizarla, definición de las redes de Petri y qué alcance tienen al modelar y resolver problemas de concurrencia, y las primitivas de sincronización.

El Estudio Preliminar tiene un conjunto de mediciones que se realizaron para determinar el alcance posible de las mejoras y la selección de la implementación a realizar.

Luego se plantea la Hipótesis del trabajo, indicando los resultados que se esperan obtener.

En Desarrollo e Implementación se explica cómo se logra implementar dicha arquitectura dentro del simulador, cómo se encuentra formada esta arquitectura y el funcionamiento de la misma.

En el apartado de Simulaciones y Mediciones se muestran las tablas y gráficas correspondientes a las simulaciones realizadas, conteniendo parámetros indicadores del rendimiento de la nueva arquitectura propuesta.

Dentro de los Anexos I, II, III y IV, se describe mas acabadamente al simulador, se detalla la instalación del simulador, como montar el simulador en un Entorno de Desarrollo Integrado (IDE), parámetros de configuración del simulador, mapeo de instrucciones, archivos de configuración, interpretación de resultados, programas (códigos) ejecutados con la información arrojada por el simulador y pruebas realizadas.

## 2 OBJETIVOS

---

Los objetivos propuestos para gestionar los recursos y esfuerzos son:

- Implementar un módulo de sincronización por hardware que permita mejorar los tiempos de sincronización entre procesos que sean ejecutados en procesadores múltiples core, múltiples hilos, empleando primitivas de sincronización por redes de Petri (ver página 25 y ss.) y además debe ser integrable al simulador SESC, cumpliendo con las restricciones listadas posteriormente.
- Medir y comparar los tiempos de sincronización y la cantidad de instrucciones ejecutadas entre procesos, en un procesador multicore (Open Sparc T1) y el mismo procesador modificado, o sea, al que se le incorpora el módulo de sincronización.

### 2.1 Restricciones

Los desarrollos a realizar para poder alcanzar los objetivos del trabajo estarán limitados por una serie de restricciones que acotan las posibles implementaciones, estas son:

- No cambiar la Arquitectura del Set de Instrucciones (ISA) del procesador.
- No requerir operaciones específicas del Sistema Operativo (SO).
- Los cambios a nivel de programa deben ser mínimos.
- Debe permitir remplazar fácilmente una primitiva de sincronización del SO.
- Los cambios a nivel de hardware, en lo posible, deben ser implementados con módulos existentes (ya probados, a los cuales solo se le debe modificar la lógica interna) y permitir así su instalación en distintos procesadores.



## 3 MARCO TEÓRICO

---

En esta sección se desarrollaran los conceptos teóricos necesarios para poder comprender la implementación realizada. Se mostrará un análisis de los simuladores existentes y se extenderán los conceptos sobre el simulador que se utilizará en el desarrollo. Luego se introducirán conocimientos sobre autómatas y Redes de Petri, como modelos de sistemas discretos para compararlos y determinar el más apto para sistemas concurrentes. Posteriormente se desarrolla la teoría de sincronización, para concluir integrando a las redes de Petri como un mecanismo formal para el modelado e implementación de sistemas concurrentes.

### 3.1 Simulación y simuladores

Para la realización de un estudio o desarrollo de mejoras sobre un sistema real uno de los mecanismos para experimentar sobre él es la simulación, esta nos permite comprender aspectos de un sistema real, al cual por diferentes motivos (económicos, de seguridad, etc.) no se puede acceder.

Una definición más formal realizada por R.E. Shannon es: "*La simulación es el proceso de diseñar un modelo de un sistema real y llevar a término experiencias con él, con la finalidad de comprender el comportamiento del sistema o evaluar nuevas estrategias (dentro de los límites impuestos por un cierto criterio o un conjunto de ellos) para el funcionamiento del sistema*" (4).

En la investigación en microarquitectura, por lo general los investigadores tienen algún tipo de propuesta de un microprocesador que será mejor que el desarrollado actualmente. Puede ser más rápido, utilizar menos energía o ser más confiable. En cualquier caso, ya que es caro para diseñar y fabricar un microprocesador, se realizan las investigaciones en simuladores de microprocesadores.

Existen diferentes enfoques para los simuladores. Algunos son impulsados por la traza, es decir, que utilizan las huellas de instrucción de las aplicaciones. Otros son impulsados por la ejecución, es decir, que en realidad ejecutan la aplicación simulada. Muchos simuladores también se dividen están formados por una parte de simulación y otra de emulación, siendo la primera un simulador que modela tiempos y energía, y la segunda la que ejecuta la aplicación simulada. (5)

Para llevar a cabo la simulación de procesadores de múltiples cores se analizaron varios simuladores:

- **LIMES**

Este simulador consiste en un entorno que emula un sistema multiprocesador con memoria centralizada. Esto significa que los sistemas que se pueden simular son sistemas basados en un único bus compartido por todos los procesadores. Al mismo tiempo, y dado que sólo existe un bus, los protocolos de caché implementados son de sondeo (snoopy).

Este simulador se ejecuta bajo Linux y consiste en un árbol de directorios donde se encuentran las fuentes del simulador. Cada vez que se realiza una simulación hay que compilar las fuentes puesto que los diferentes parámetros de simulación se encuentran en las propias fuentes.

- **SESC**

SESC es un simulador de arquitectura de microprocesadores desarrollado por el grupo de investigación I-ACOMA de la Universidad de Illinois.

El simulador modela diferentes tipos de arquitecturas: single processors, symmetric multiprocessor (SMP), common memory processor (CMP), processing in memory (PIM) y thread level speculation (TLS).

SESC puede ser ejecutado bajo plataforma Linux. El código fuente del simulador esta disponible en repositorios de sourforge, pero la documentación del mismo es escaso.

- **SMPCache**

SMPCache es un simulador mediante trazas para el análisis y la docencia de sistemas de memoria caché en multiprocesadores simétricos. La simulación se apoya en un modelo construido de acuerdo con los principios básicos arquitectónicos de estos sistemas. El simulador posee una interfaz gráfica completa y amigable, y opera en sistemas PC con Windows. Posee solo un nivel de cache.

- **Simics**

Simics es un simulador full-system usado para ejecutar archivos fuentes sin cambios para el hardware destino a velocidades de alta performance. Simics fue originalmente desarrollado por el Instituto Sueco de Ciencias de la Computación (SICS), y luego se separó a Virtutech para el desarrollo comercial en 1998. Virtutech fue adquirida por Intel en 2010 y Simics se comercializa a través de una subsidiaria, Wind River.

Las arquitecturas que se pueden simular son: Alpha, x86-64, IA-64, ARM, MIPS, Power-PC, SPARC-V8/9 y x86. El propósito de la simulación con Simics es desarrollar software para tipos particulares de hardware embebido, utilizando al simulador como una plataforma virtual.

Simics esta disponible para plataformas Windows y Linux.

A continuación se presenta una tabla donde se comparan diferentes características de los simuladores anteriormente descriptos.

	<b>LIMES</b>	<b>SESC</b>	<b>SMPCache</b>	<b>Simics</b>
<b>SO</b>	Linux	Linux	Windows	Linux Windows
<b>Open Source</b>	Si cumple	Si cumple	No cumple	Si cumple
<b>Tipo de</b>	Software Libre	Software Libre	Freeware	Software

	<b>LIMES</b>	<b>SESC</b>	<b>SMP Cache</b>	<b>Simics</b>
<b>licencia</b>				Propietario
<b>Documentacion disponible</b>	Media	Baja	Media	Alta
<b>Lenguaje</b>	C++	C++	Desconocido	Python
<b>Niveles de cache</b>	Uno	Ilimitado	Uno	Dos

**Tabla 1. Tabla comparativa de Simuladores**

A partir de analizar y estudiar las distintas características de los simuladores que se muestran en la Tabla 1, se resolvió que solo se podían emplear LIMES o SESC, ya que son del tipo Software Libre, escritos en C++, para Linux y existe alguna documentación explicativa de estos simuladores. Posteriormente se decidió utilizar SESC, ya que este posee más niveles de cache, siendo así el más apto para llevar a cabo el desarrollo del proyecto.

### 3.1.1 SESC (5)

SESC es un simulador cuya sigla significa: **S**imulador **S**uper**E**SCalar. Se inicia como un proyecto “mascota” de José Renau mientras realizaba un PhD en la Universidad de Illinois (Urbana-Campaing) en el grupo I-ACOMA.

SESC modela un gran número de arquitecturas: single processors, SMP (Symmetric Multiprocessor), CMPs (Comun Memory Processor), PIMs (Processing In Memory) y Thread Level Speculation (TLS). Siendo la utilizada en el desarrollo del trabajo la SMP. Se trata de un tipo de arquitectura de ordenadores en que dos o más procesadores comparten una única memoria central.

#### 3.1.1.1 ¿Cómo funciona SESC?

SESC está dividido en 2 partes bien definidas. La primera es un emulador denominado MINT, este interpreta las instrucciones MIPS para ejecutarlas sobre la arquitectura donde está corriendo SESC. La segunda es la que realiza la contabilidad de tiempos, denominado simulador de tiempos, este tiene en cuenta los recursos necesarios por cada instrucción a ejecutar, entendiéndose por recursos a los registros, posiciones de memoria y unidades funcionales (unidad aritmética y lógica donde se ejecutan las operaciones, tanto enteras como de punto flotante). El simulador de tiempos planifica las instrucciones y recursos para determinar los tiempos de procesador insumidos por la traza ejecutada.

A continuación se introducirá la secuencia de pasos para simular un programa en SESC. Al seguir el orden de pasos descriptos es conveniente ir revisando la Figura 1.

Como primer paso para simular se debe escribir el programa en lenguaje C o C++, teniendo en cuenta las instrucciones soportadas por el simulador (ver mapeo de instrucciones en Anexo II página 90). Una vez que tenemos nuestro código fuente completo debemos compilarlo para la arquitectura MIPS, esto se realiza con la herramienta de compilación cruzada SESCUtils (ver pág. 85 en Anexo II). Una vez que tenemos nuestro archivo ejecutable estamos ya en condiciones de correrlo dentro de SESC.

El simulador acepta distintos parámetros que podemos ingresarle al momento de ejecución. Uno de los parámetros más importantes es el del archivo de configuración a utilizar, el cual contendrá todas las configuraciones correspondientes al procesador (o procesadores) a simular junto con la jerarquía de memoria (Anexo III).

Una vez finalizada la ejecución del programa se obtienen el registro de tiempo generado por el simulador para ser analizado. SESC nos provee de un script para simplificar la lectura de los resultados de la simulación (ver Interpretación de los resultados de una simulación en pág. 103).

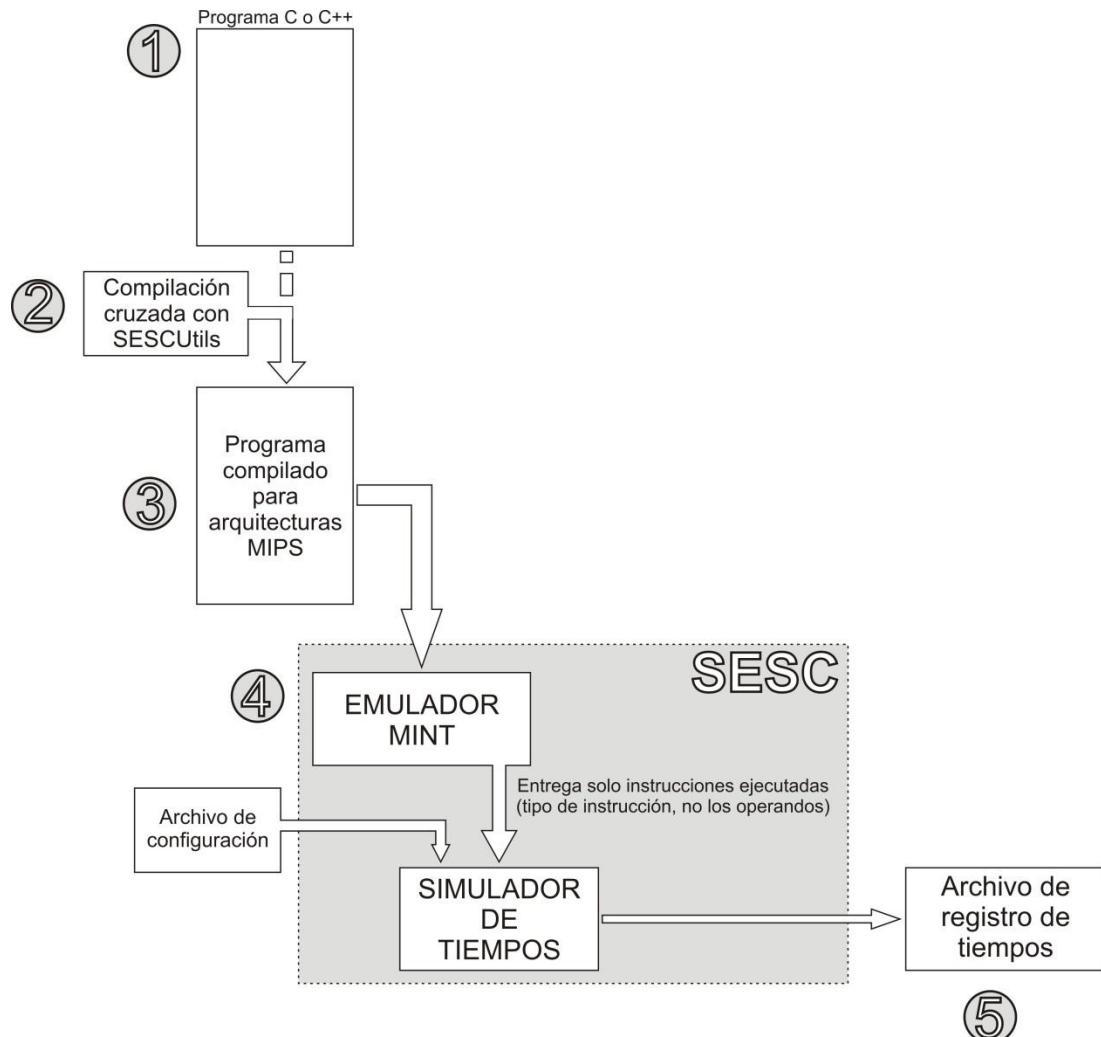


Figura 1. Secuencia de pasos para simular en SESC

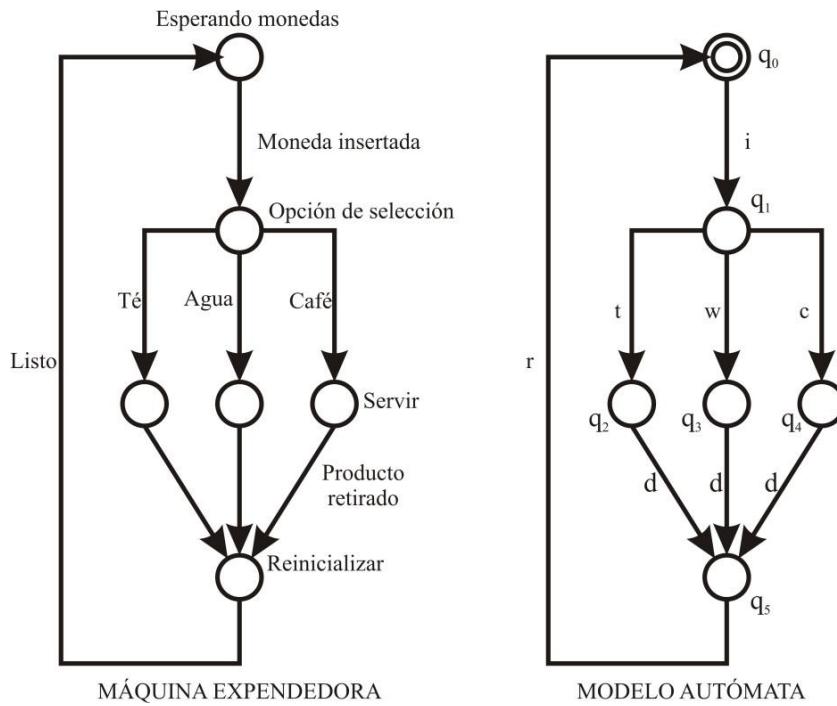
## 3.2 Modelos

Los sistemas de eventos discretos (DES) son sistemas dinámicamente guiados por la ocurrencia de eventos. Los autómatas y las Redes de Petri denotan las dos mayores clases de modelos DES (3). En los siguientes apartados introduciremos los conceptos sobre autómatas y mas exhaustivamente de PN, para concluir con la comparación de ambos modelos y la selección del más apto para nuestro trabajo.

### 3.2.1 Autómatas

Formalmente, un autómata determinista (usualmente llamado máquina de estado) es una tupla de cinco elementos  $G = (Q, \Sigma, \delta, s, F)$ , donde  $Q$  denota un conjunto de estados,  $\Sigma$  un conjunto de eventos,  $\delta$  las funciones de transición,  $s$  el estado inicial y  $F$  un conjunto de estados finales.

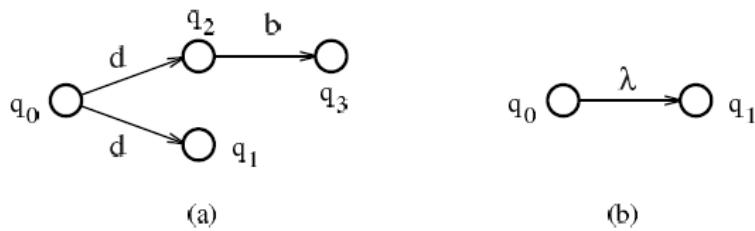
El comportamiento de los sistemas modelados por los autómatas puede ser caracterizado por la secuencia de eventos posibles en el autómata, comenzando en el estado inicial. Como denota la Figura 2 la secuencia  $\langle r, i, t \rangle$  es posible, pero no  $\langle r, t, i \rangle$  ya que el autómata no puede ejecutar  $t$  desde el estado  $q_0$ . Podemos también incluir al conjunto de eventos al evento nulo ( $\lambda$ ), el cual denota una transición que puede ocurrir en cualquier momento, independientemente de la ocurrencia de un evento (ver Figura 3b).



**Figura 2. Automata de máquina expendedora**  
Fuente: Supervisory control of concurrent systems (6)

El *lenguaje* de un autómata determinista  $G$  se define como  $L(G) = \{\sigma \in \Sigma^* : \delta(s, \sigma) \text{ es definida}\}$ .  $\Sigma^*$  incluye al evento nulo.

Notar que se hace siempre referencia a autómatas deterministas. Los autómatas no deterministas extienden a los deterministas permitiendo transiciones diferentes desde un mismo estado producidas por el mismo evento, esto quiere decir que la elección de la transición hacia alguno de los posibles estados va a estar dada, por ejemplo, aleatoriamente. Ese tipo de transición se puede ver claramente en la Figura 3a. Pueden también contener el evento nulo.



**Figura 3. Extensiones a los autómatas**  
Fuente: Supervisory control of concurrent systems (6)

Los autómatas no deterministas se definen con la tupla de manera similar a los deterministas, salvo  $\delta : Q \times (\Sigma \cup \{\lambda\}) \rightarrow 2^Q$ , donde  $2^Q$  denota subconjuntos de  $Q$ .

### 3.2.2 Redes de PETRI (7)

Una Red de Petri o Petri Net (PN) es un modelo gráfico, formal y abstracto para describir y analizar el flujo de información. Conforma una herramienta matemática que puede aplicarse especialmente a los sistemas paralelos que requieran simulación y modelado de la concurrencia en los recursos compartidos. Las PN están asociadas con la teoría de grafos y se pueden considerar como autómatas formales y generadores de lenguajes formales.

Los siguientes componentes conforman una PN:

- *Lugares*: graficados por un círculo, permiten representar estados del sistema. Más precisamente los lugares juegan el papel de variables de estado y toman valores enteros.
- *Token*: representan el valor específico de un estado o lugar, interpretándose como recursos. El número y su ubicación en los lugares varían dinámicamente en función de la ejecución de la red.
- *Transiciones*: graficadas por segmentos rectos, representan el conjunto de sucesos que provocan la modificación de los estados del sistema.
- *Arcos dirigidos*: indican las conexiones entre lugares y transiciones. Nunca unen dos lugares o dos transiciones en forma sucesiva. Cabe aclarar que pueden entrar o salir varios arcos de una misma transición o de un mismo lugar.
- *Peso*: número entero asociado a cada arco, que indica la cantidad de tokens que se consumen o generan al atravesarlo. El disparo de una transición retira tantos tokens de cada uno de sus lugares de entrada como lo indican los pesos de los arcos conectores y añade los tokens a sus lugares de salida como lo indican los pesos de los arcos de salida. Los arcos que no especifican un peso, se suponen con peso unitario.

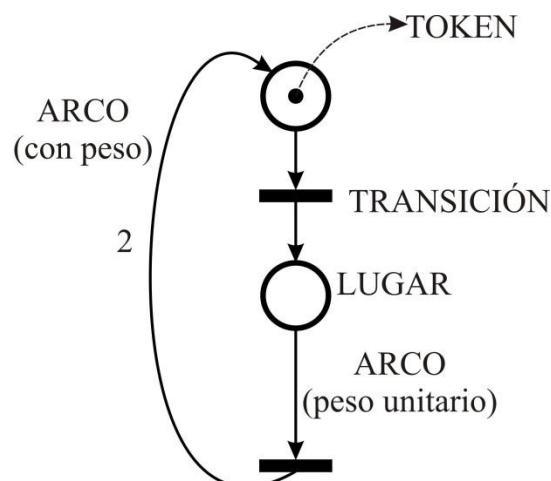


Figura 4. Elementos de una Red de Petri

#### 3.2.2.1 Definiciones Matemáticas de las Redes de PETRI

Una PN se compone de los siguientes elementos:

- $L = \{l_1, l_2, l_3 \dots l_m\}$ , Conjunto de m lugares con m finito y distinto de cero;
- $T = \{t_1, t_2, t_3 \dots t_n\}$ , Conjunto de n transiciones con n finito y distinto de cero.

- $I^-$ , matriz de incidencia negativa. Esta matriz es de dimensiones  $m \times n$  y representa los pesos de los arcos que ingresan desde los lugares de  $L$  a las transiciones de  $T$ .
- $I^+$ , matriz de incidencia positiva. Esta matriz es de dimensiones  $m \times n$  y representa los pesos de los arcos que salen desde las transiciones de  $T$  hacia los lugares de  $L$ ;

A partir de las dos últimas definiciones, se obtiene la Matriz de Incidencia:

$$I = I^+ - I^-$$

Para una red con  $m$  lugares y  $n$  transiciones, la misma es una matriz  $m \times n$  cuyos elementos  $a_{ij}$  son:

$\forall l_i \in L \text{ y } \forall t_j \in T$  se tiene:

- $a_{ij} = 0$ , si entre  $l_i$  y  $t_j$  no hay arcos que los relacionen;
- $a_{ij} = W_{ij}$ , si  $l_i$  es salida de  $t_j$
- $a_{ij} = -W_{ij}$ , si  $l_i$  es entrada a  $t_j$

Con  $W_{ij}$  peso del arco que une  $l_i$  con  $t_j$

A continuación se explican las distintas propiedades de las PN.

**Marcado:** es la distribución de los tokens en los lugares. Se corresponde con estados específicos de la red. Este varía con la ejecución del sistema, lo que permite representar la distribución de los recursos en los lugares a medida que transcurre el tiempo. Es un vector de dimensiones  $m \times 1$  siendo  $m$  la cantidad de elementos del conjunto  $L$ . Se tiene para un lugar  $l_i$  en un instante de tiempo el número de tokens como  $m(l_i)$ .

Considerando los conceptos expuestos, una Red de Petri queda definida como una 5-tupla  $PN = (L, T, I^-, I^+, m_0)$ , con  $m_0$  el marcado inicial de la red.

Se utilizan las siguientes definiciones:

- $t^-$  = conjunto de lugares de entrada a  $t$ ;
- $t^+$  = conjunto de lugares de salidas de  $t$ ;
- $l^-$  = conjunto de transiciones de entrada de  $l$ ;
- $l^+$  = conjunto de transiciones de salida a  $l$ .

**Transición preparada:** se dice que  $t_i$  lo está si y sólo si todos sus lugares de entrada tienen al menos la cantidad de tokens igual al peso de los arcos que los une a  $t_i$ . Esto es  $\forall l_i \in I^-(t_j) \Rightarrow m(l_i) \geq W_{ij}$  con  $W_{ij}$  peso del arco que une  $l_i$  con  $t_j$ . Si dos o más transiciones se encuentran preparadas en un mismo instante, se dice que están preparadas concurrentemente.

**Disparo de una transición:** ocasiona el cambio de estado de una red, es decir el cambio del marcado de la misma.

La función de disparo  $\partial$  para una transición preparada  $t_i$  establece:

$$\partial(m_k, t_j) = \begin{cases} m_{k+1}(l_i) = m_k(l_i) - W_{ij} & \forall l_i \in \bullet t_j; \\ m_{k+1}(l_i) = m_k(l_i) + W_{ij} & \forall l_i \in t_j \bullet; \\ m_{k+1}(l_i) = m_k(l_i) & \text{en otro caso.} \end{cases}$$

**Ejecución de una PN:** es la secuencia de pasos que resultan de disparar la red n veces partiendo desde el marcado inicial  $m_0$ , la que puede representarse mediante dos sucesiones:

- Una secuencia de marcados por los que pasa la red en forma secuencial ( $m_0, m_1, \dots, m_n$ );
- Una sucesión de transiciones que se van disparando ( $t_0, t_1, \dots, t_n$ ) tales que  $\partial(m_k, t_i) = m_{k+1}$ .

La ejecución de una red no es única, dando así una característica de no determinismo.

- **Persistencia de una PN:** dadas dos transiciones habilitadas de la PN, el disparo de una de las mismas no deshabilita a la otra. Una transición habilitada en una red persistente permanecerá en ese estado hasta que se dispare.
- **Ecuación de Estado:** la ecuación de estado de una red de Petri (que explicita el estado de la red en cada instante), queda definida en función de la matriz de incidencia  $I$ , y de un vector de disparo  $\sigma$  de dimensión  $1 \times n$  (siendo  $n$  la cantidad de transiciones) cuyas componentes son todas nulas, excepto la que corresponde a la transición disparada en el instante  $i$ , que vale 1:

$$m_{i+1} = m_i + I \times \sigma_i$$

Se deduce que el marcado final de una secuencia de disparos (partiendo de del marcado inicial  $m_0$ ), puede obtenerse aplicando sucesivamente la ecuación de estado, quedando así lo siguiente:

$$m_{i+1} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \times \sigma$$

$$\sigma = [1 \ 0 \ 0 \ 0] \Rightarrow m_{i+1} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\sigma = [0 \ 0 \ 0 \ 1] \Rightarrow m_{i+1} = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 2 \\ 0 \\ -1 \end{bmatrix}$$

$$m_{i+1} = m_i + I \times \sigma$$

Con  $\bar{s} = \sum_{j=1}^i \sigma_j$  vector asociado a la secuencia de disparo de las transiciones de la red; donde su  $j$ -ésima componente es igual al número de veces que la transición  $t_j$  se ha disparado.

Dada una red de Petri, un marcado  $m'$  es dinámicamente alcanzable desde el marcado  $m_1$  si y solo si existe una sucesión finita de marcados  $m_1, m_2, \dots, m_n$  y de transiciones  $t_1, t_2, \dots, t_n$  tales que permitan alcanzar  $m_n$  a partir de  $m_1$ . El conjunto de marcados dinámicamente alcanzables viene definido por  $\text{mark}(PN)$ . Matemáticamente, está dado por la clausura reflexiva y transitiva de la relación de equivalencia de alcanzabilidad inmediata.

- **Alcanzabilidad de un marcado en una PN:** dada una PN y considerando un marcado  $m$ , se desea determinar si  $m$  es alcanzable desde el marcado inicial  $m_0$ , es decir verificar si  $m \in \text{mark}(PN)$ .

Una forma de resolver el problema de alcanzabilidad es plantear el sistema de ecuaciones derivado de la ecuación de estado de la red:

$$m = m_0 + I * X; \text{ donde } X = [x_1, x_2, \dots, x_n]^T \text{ (incógnitas) si la red tiene } n \text{ transiciones.}$$

Si el sistema de ecuaciones planteado tiene solución, existe una secuencia de disparos que permite alcanzar  $m$ . Cabe aclarar que esto no explica el orden en que deben dispararse las transiciones. Es importante saber que:

- Si el sistema de ecuaciones no tiene soluciones el marcado es inalcanzable;
- Si el sistema admite más de una solución, hay varias secuencias de disparo posibles, pero dado que la red no es determinista, no se puede especificar cuál de ellas se dará.

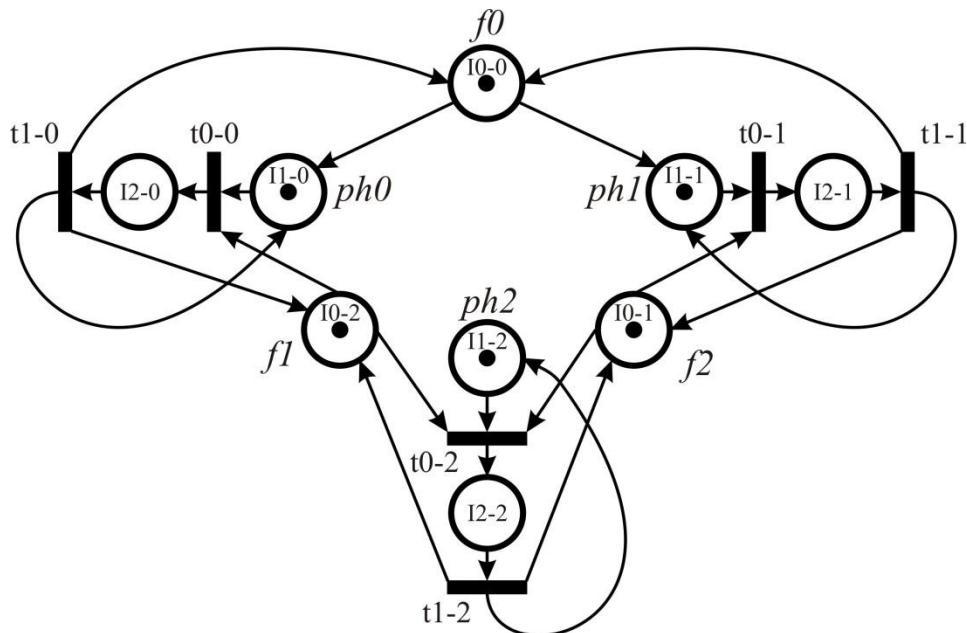
*Para dejar mas claro los conceptos expuestos con anterioridad se va a plantear un ejemplo.*

### 3.2.2.1.1 Ejemplo: Red de Petri, 3 FILÓSOFOS COMENSALES (8)

El algoritmo de los tres filósofos es un problema común para evidenciar la concurrencia. Esta definido por tres filósofos que dedican sus vidas a pensar y comer (acciones finitas en el tiempo). Los filósofos comparten una mesa rodeada de tres sillas, cada una de un filósofo. En el centro de la mesa hay comida, y en la mesa tres palitos y tres platos. Cuando un filósofo no se relaciona con sus colegas se supone que esta pensando. De vez en cuando, un filosofo siente hambre, este se dirige a su silla, trata de tomar los dos palitos que están mas cerca de el para comer. Cuando el filósofo tiene sus dos palitos simultánea-

mente, come sin dejar los palitos. Cuando ha acabado de comer, vuelve a dejar los dos palitos en la mesa y empieza a pensar de nuevo. La solución a este problema consiste en inventar un algoritmo que permita comer a todos los filósofos, evitando que alguno muera de hambre (aquel que nunca puede tomar los dos palitos para comer).

A continuación (Figura 5) se muestra cómo se plantea la red de los “tres filósofos Chinos”. Esta red tiene todos sus arcos de peso uno, por ello se omiten dichos pesos.



**Figura 5. Red de Petri de Los Filósofos Chinos**  
Fuente: Programación Concurrente (8)

La matriz de incidencia y el marcado de la red son los siguientes:

$$I = \begin{array}{|c c c c|c c c|c|c|} \hline & t0-0 & t0-1 & t0-2 & t1-0 & t1-1 & t1-2 & & \\ \hline \text{f0} & -1 & -1 & 0 & 1 & 1 & 0 & I0-0 & 1 \\ \text{f1} & 0 & -1 & -1 & 0 & 1 & 1 & I0-1 & 1 \\ \text{f2} & -1 & 0 & -1 & 1 & 0 & 1 & I0-2 & 1 \\ \hline \text{p0} & -1 & 0 & 0 & 1 & 0 & 0 & I1-0 & 1 \\ \text{p1} & 0 & -1 & 0 & 0 & 1 & 0 & I1-1 & 1 \\ \text{p2} & 0 & 0 & -1 & 0 & 0 & 1 & I1-2 & 1 \\ \hline \text{I0-0} & 1 & 1 & 0 & -1 & 0 & 0 & I2-0 & 0 \\ \text{I0-1} & 1 & 0 & -1 & 0 & -1 & 0 & I2-1 & 0 \\ \text{I0-2} & 0 & 1 & 0 & 0 & -1 & 0 & I2-2 & 0 \\ \hline \end{array} \quad m_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

En la Figura 5 se observa:

1. ph0, ph1, ph2; filósofos, cuyos platos son I1-0, I1-1, I1-2
2. I0-0, I0-1, I0-2; son los “palitos” para poder comer
3. t0-0, t0-1, t0-2; Son las transiciones “comer”
4. I2-0, I2-1, I2-2; Es el estado comiendo

5. t1-0, t1-1, t1-2; Transiciones que devuelven el “filósofo” y los “palitos”

Considerando la situación en la que el filósofo 1 desea comer, se tiene un vector de disparo  $\sigma$ :

$$\sigma = [0 \ 1 \ 0 \ 0 \ 0 \ 0]$$

Haciendo  $I \times \sigma$ , se selecciona la columna:

$$I \times \sigma = \begin{bmatrix} -1 & | & l0-0 \\ -1 & | & l0-1 \\ 0 & | & l0-2 \\ 0 & | & l1-0 \\ -1 & | & l1-1 \\ 0 & | & l1-2 \\ 0 & | & l2-0 \\ 1 & | & l2-1 \\ 0 & | & l2-2 \end{bmatrix}$$

Luego el marcado resultante de aplicar la ecuación de estado  $m_1$  es el siguiente:

$$m_1 = m_0 + I \times \sigma = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Otra forma de obtener la matriz de Incidencia  $I$ , y el vector de marcado inicial  $m_0$ , es mediante el empleo de algún software de Redes de Petri, el que se utilizó en este trabajo fue Pipe, el cual nos permite obtener las matrices a partir del grafo de la red. Pipe es un programa gratuito que funciona bajo Windows y Linux.

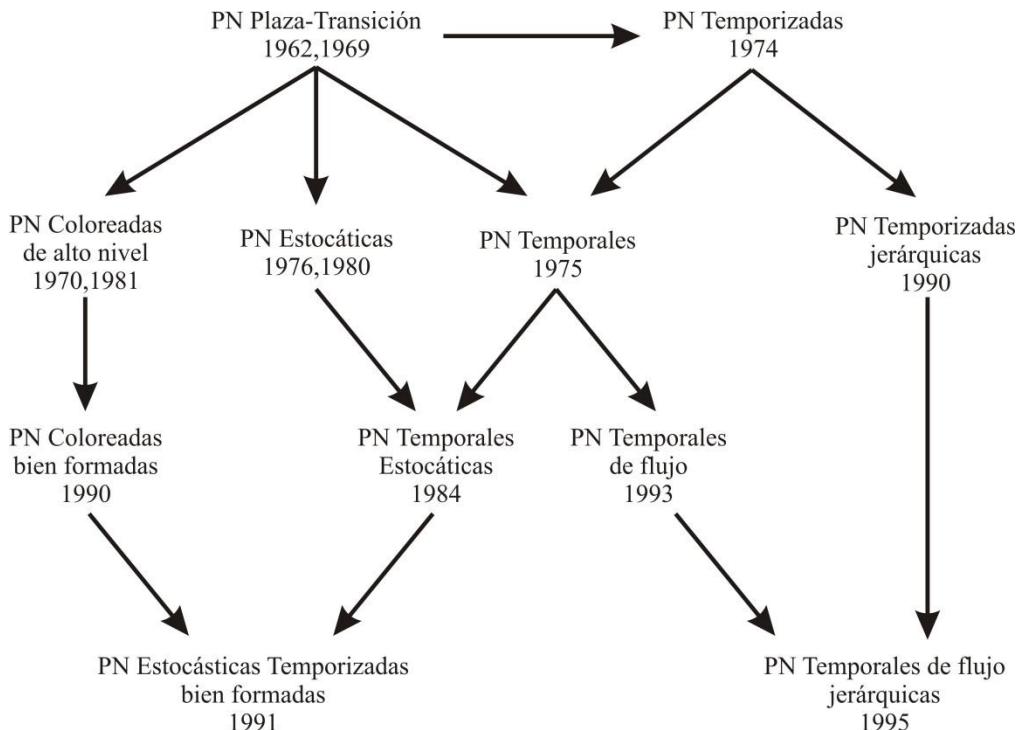
### 3.2.2.2 Familias de PN

Las PN están agrupadas según su poder de expresividad. La Figura 6 representa a las principales familias de modelos de PN cronológicamente. En la figura, las flechas apuntan hacia el modelo planteado posteriormente, o sea, el modelo al inicio de la flecha es temporalmente más antiguo que el de su finalización. Esto nos da una idea de los pasos seguidos en la investigación y desarrollo de los principales modelos de PN.

Las distintas PN se agrupan en 3 diferentes modelos que se toman como referencia, PN propiamente dichas, PN Temporales (TPN) y PN estocásticas (SPN). Cada modelo es una extensión de uno anterior, pudiendo ser simplificados para transformarse en un modelo básico de PN (3).

Cuando se aplican las PN mencionadas en el párrafo anterior al modelado de sistemas, se hace evidente que los modelos no tienen el mismo poder de aplicación, en términos de:

- La definición y descripción de conceptos de paralelismo, distribución y sincronización;
- La comprensión y uso de semántica temporal y estocástica;
- El análisis de los posibles mecanismos y comportamientos, entre diferentes contextos y aplicaciones.



**Figura 6. Cronología de las familias de PN**

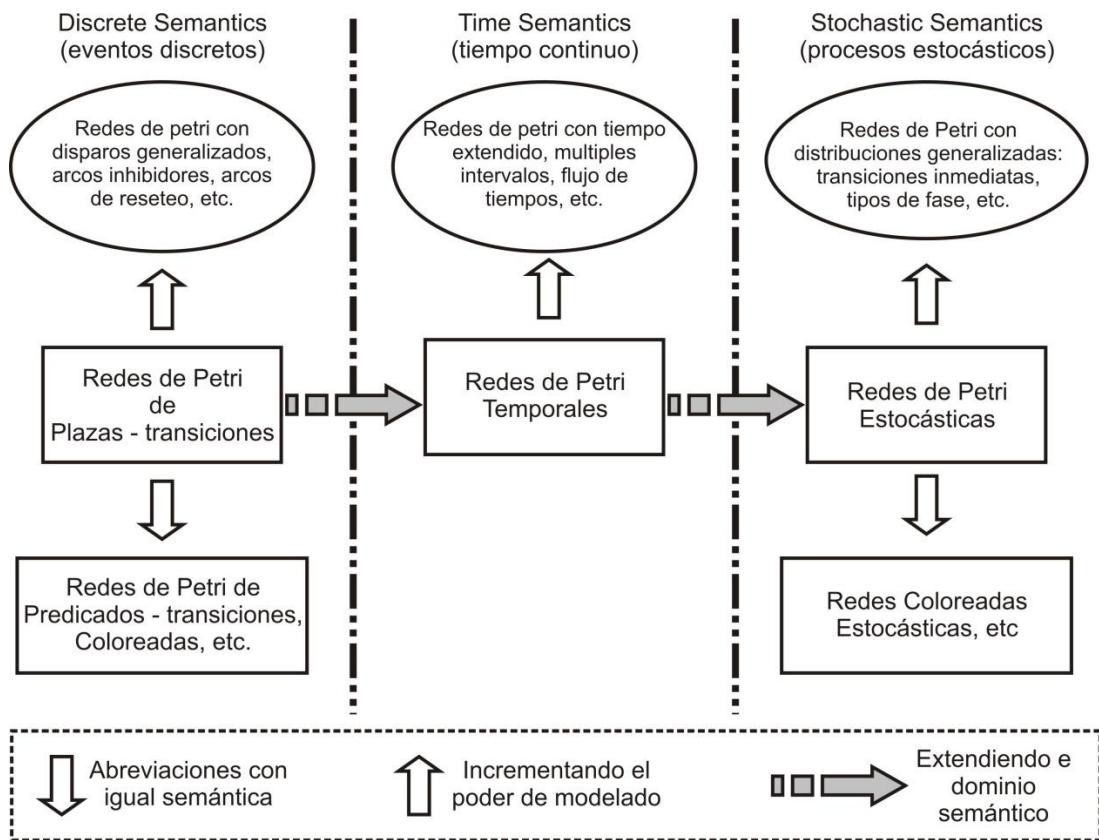
Fuente: Petri Nets: Fundamental Models, Verification and Applications (3)

La Figura 7 da una vista conceptual de los modelos mencionados anteriormente, clarificando su relación sintáctica y semántica. En la figura, tres campos son respectivamente definidos por:

- Una *semántica de estado discreto*, para redes que no tienen comportamiento temporal ni estocástico, los comportamientos son representados por una gráfica finita de todos los estados del modelo.
- La *semántica en tiempo continuo*, para extender comportamientos basados en modelos de tiempo denso (9).
- Y una *semántica estocástica*, para comportamientos que incluyen distribuciones de probabilidad.

Como vemos en la Figura 7, al partir de los modelos de referencia podemos tomar dos rumbos:

- Abreviar, simplificando el modelo y las especificaciones del sistema.
- Extender, aumentando el poder de expresividad, por ejemplo introduciendo parámetros de tiempo o distribuciones estocásticas.



**Figura 7. Dominios de las semánticas de los modelos basados en PN**  
**Fuente: Petri Nets: Fundamental Models, Verification and Applications (3)**

### 3.2.2.3 Lenguaje de las redes de Petri

La introducción de las redes de Petri extendidas tiene como objetivo incrementar el poder de expresividad de la red, mientras preservan el poder de decisión de algunas propiedades. Inicialmente, los lenguajes formales fueron estudiados en relación a la gramática (10).

Dependiendo de la estructura de la gramática se definen familias de lenguajes y problemas de estudio como:

- Los miembros de una palabra en el lenguaje.
- El elemento nulo del lenguaje
- La clausura de una familia de lenguajes bajo operaciones como unión, intersección y complemento.

Cada problema tiene una interpretación en relación al comportamiento de los sistemas modelados, por ejemplo, las redes de Petri. *El primer problema se relaciona con la prueba: donde secuencias de comportamiento esperadas realmente ocurren. El problema del elemento nulo se relaciona con la existencia de al menos una secuencia fallida. La clausura de una familia por operaciones ofrece al diseñador la posibilidad de crear sistemas modulares usando especificaciones dadas por las operaciones.* Por ejemplo, la intersección de lenguajes corresponde muy usualmente a la sincronización entre subsistemas.

El estudio realizado de los problemas en *Petri Nets: Fundamental Models, Verification and Applications* - Cap. 4.7 (3) arroja algunas definiciones importantes sobre las Redes de Petri:

- Los lenguajes de Redes de Petri son cerrados por unión e intersección.

- Los problemas de miembros y del elemento nulo son determinables por lenguajes de Redes de Petri.
- La familia de lenguajes de PN contiene a los lenguajes regulares y no es comparable con la familia de lenguajes algebraicos. (11)

Podemos concluir que las redes de Petri no solo nos proveen de modelos para representar procesos concurrentes, sino que poseen una semántica propia de los lenguajes formales.

### 3.2.3 PN vs. Autómata (6)

Anteriormente vimos que la transición de un autómata está indicada por eventos, pudiendo el mismo evento denotar dos o más transiciones diferentes (autómata no determinista). Las PN en los cuales las transiciones están etiquetadas por eventos se denominan PN etiquetadas. Este tipo de redes le agregan a las PN una función de etiquetado definida por el conjunto de eventos, incluyendo al evento nulo. Cuando cada transición está definida por un único evento (no existen dos transiciones con la misma etiqueta) las PN se denominan no etiquetadas.

***La PN pueden modelar los autómatas a través del grafo de alcanzabilidad (6) de una red.*** Cuando se plantea el grafo de alcanzabilidad de una red puede ser directamente visto como un autómata, en general corresponde a autómatas no deterministas.

Los autómatas finitos pueden ser fácilmente convertidos a PN etiquetadas, resultando estas PN en máquinas de estado.

Comparada con los autómatas, las PN ofrecen una representación más compacta de sistemas concurrentes. Además, el modelado de estos sistemas es más natural usando PN.

### 3.2.4 Seleccionando un modelo

En general, el diseñador debe tener un acabado entendimiento de la semántica fundamental de un sistema. Así, para arquitecturas simples, el modelo puede ser capaz de representar de manera fiel todos los detalles del sistema. Sin embargo, para sistemas complejos, generalmente será imposible, por razones económicas, representar el detalle de todas las funciones existentes, y será necesario seleccionar y validar ciertos bloques construidos.

De los conceptos vertidos en esta sección, las PN y sus extensiones son de un interés fundamental innegable, porque:

- Proveen el primer modelo aproximado por la semántica de los sistemas concurrentes;
- Define un soporte gráfico fácil para la representación y el entendimiento de mecanismos y comportamientos básicos de sistemas concurrentes;
- Prueba ser, iniciando desde las máquinas de estado, una fácil extensión de aproximaciones previas, y al mismo tiempo, la creación, el análisis y la ejecución de modelos;
- Expresa muy simplemente los conceptos básicos en comunicación, incluyendo espera y sincronización, y además tiene en cuenta los parámetros temporales y estocásticos;
- Asegura mantenerse alejado de cualquier lenguaje particular de implementación.

Por los motivos listados anteriormente, y teniendo en cuenta que las redes de Petri aparte de proveernos de un mecanismo formal para modelar procesos concurrentes son

en si mismas un lenguaje formal (3), seleccionamos a las PN para modelar y probar los sistemas estudiados durante el proyecto.

### 3.3 Teoría de Sincronización

En esta sección se definen los conceptos de dos tipos de sincronización entre procesos concurrentes, exclusión mutua y condición de sincronización.

#### 3.3.1 Exclusión Mutua

Cuando los procesos desean utilizar un recurso no compartible, la sincronización necesaria entre ellos se denomina *Exclusión Mutua*. Un proceso que está accediendo a un recurso no compartible se dice que se encuentra en su sección crítica. Por lo tanto la sección crítica es la parte del código que utiliza un proceso en el acceso a un recurso no compartible y, por tanto debe ejecutarse en exclusión mutua. Cuando un proceso está en su sección crítica, el resto de los procesos (o al menos aquellos que tengan acceso al mismo recurso no compartible) no deben encontrarse en sus secciones críticas, y si quieren acceder a ellas deben quedarse a la espera de que la sección crítica quede libre. Cuando un proceso termina de ejecutar su sección crítica, entonces se debe permitir a otro proceso que se encontraba esperando, la posibilidad de entrar a su sección crítica.

#### 3.3.2 Condición de Sincronización

La condición de sincronización hace referencia a la situación en la cual es necesario coordinar la ejecución de procesos concurrentes, la cual sucede cuando un objeto de datos compartido está en un estado inapropiado para ejecutar una operación determinada. Cualquier proceso que intente realizar esta operación sobre el objeto, debe ser retrasado hasta que el estado del objeto cambie por las operaciones de otros procesos. Por lo tanto *se define la condición de sincronización como la propiedad requerida de que un proceso no realice un evento hasta que otro proceso haya emprendido una acción determinada*.

#### 3.3.3 Solución a los tipos de Sincronización

A continuación se presentan los distintos y más importantes mecanismos existentes para resolver e implementar la sincronización necesaria entre procesos concurrentes, estos son:

- *Inhibición de las interrupciones*
- *Espera Ocupada*
- *Semáforos*
- *Regiones Críticas*
- *Monitores*
- *Operaciones de paso de mensaje*

Haciendo foco especialmente en los mecanismos de espera ocupada, semáforos, regiones críticas y monitores los cuales pertenecen al grupo de soluciones basadas en variables compartidas, el resto de los mecanismos pertenecen al grupo de soluciones basadas en paso de mensaje. Este último grupo no es relevante ya que posee su principal aplicación en sistemas de memoria distribuidos, los cuales no están incluidos en las arquitecturas sobre las que se va realizar el desarrollo.

### 3.3.3.1 Espera Ocupada

El término espera ocupada hace referencia a que esta implementación de la sincronización se basa en que un proceso espera mediante la comprobación continua del valor de una variable manteniendo ocupada la CPU. Según el tipo de operaciones que se utilicen para desarrollar una solución basada en espera ocupada podemos distinguir:

1. **Soluciones de Software:** Las únicas operaciones atómicas que se consideran son las instrucciones de bajo nivel para leer y almacenar de/en direcciones de memoria. Es decir si dos instrucciones de este tipo se produjeran simultáneamente, el resultado equivaldría a la ejecución secuencial en un orden desconocido.
2. **Soluciones de Hardware:** Se utilizan instrucciones especializadas que llevan a cabo una serie de acciones de forma invisible como leer y escribir, intercambiar el contenido de dos posiciones de memoria, etc.

En este tipo de soluciones, para indicar una condición, un proceso inicializa el valor de una variable compartida; para esperar una condición, un proceso comprueba repetidamente dicha variable hasta que toma el valor deseado.

### 3.3.3.2 Semáforo

La solución planteada con anterioridad tiene la gran desventaja de que el proceso espera de forma activa, lo cual limita la eficiencia del sistema ya que no está haciendo nada y ocupa el procesador. La primitiva de comunicación entre procesos que denominamos semáforo bloquea al proceso en lugar de desperdiciar tiempo de CPU cuando no se le permite entrar a su sección crítica o realizar una acción determinada. Pudiendo ser implementado a nivel de software, o a nivel de hardware con una serie de instrucciones que no todos los procesadores poseen.

Podemos considerar que un semáforo es un tipo de datos abstracto que por regla general, solo admite como posibles valores enteros no negativos. Si el semáforo solo admite como valores 0 y 1, se denomina semáforo binario. En cualquier otro caso estaremos hablando de un semáforo general. Básicamente las operaciones que se pueden realizar sobre los semáforos son:

- Wait (s). Decrementa el valor de s si es mayor que cero. Si s=0, el proceso se bloquea en el semáforo.
- Signal (s). Desbloquea algún proceso bloqueado en s, y en el caso de que no haya ningún proceso, incrementa el valor de s.
- Initial (s,v). Se utiliza para inicializar el semáforo s al valor v donde  $v \geq 0$ . Esta operación es exclusiva para Pascal-fc.

<pre>wait (s) If s&gt;0 then     s = s - 1 else bloquear proceso</pre>	<pre>signal (s) if (hay procesos bloqueados)     desbloquear un proceso else s = s + 1</pre>
--	--

Como puede apreciarse en el pseudocódigo anterior, tanto la operación *wait* como la operación *signal* implican una comparación y una actualización de la variable entera

asociada o, en su caso, el bloqueo o desbloqueo de los procesos. Una de las principales características de los semáforos es que todas las operaciones asociadas al *wait* y al *signal* deben ejecutarse de forma indivisible, es decir, la utilización de un semáforo implica la exclusión mutua en la ejecución de las operaciones, sobre él definida. Los semáforos usan de las funciones proporcionadas por el SO para bloquear a los procesos, provocando una espera con bloqueo. Dicho empleo de funciones proporcionadas por el SO tiene asociado un gran costo de cálculo, ya que todo el semáforo requiere gran cantidad de ciclos de reloj de CPU (8). En el estudio preliminar (página 27) se puede apreciar que un semáforo insume 145 ciclos de reloj y ejecuta decenas de instrucciones, muy superior a la escritura o lectura de una variable.

### 3.3.3.3 Sección Crítica

La sección crítica es un mecanismo que permite tratar de manera diferente los problemas de sincronización y los problemas de acceso a la sección crítica. Este mecanismo da la posibilidad de diseñar programas en los que sería bastante sencillo diferenciar las partes del código que se utilizan para controlar el acceso a la sección crítica de las que resuelven los problemas de sincronización.

Definimos a la sección Critica Condicional (RCC) como un mecanismo estructurado que va a permitir diferenciar el control de acceso a la sección crítica, de la implementación de las condiciones de sincronización. El uso de RCC hace que el compilador imponga determinadas restricciones al uso de variables compartidas, es decir el acceso indebido a las variables compartidas va a ser detectado en tiempo de compilación. Esta propiedad llevará a diseñar programas estructurados y fáciles de mantener.

Para detectar el uso incorrecto de variables compartida, estas tienen que ser declaradas de una forma especial. Para mayor información consulte (8).

### 3.3.3.4 Monitores

Los monitores son mecanismo de abstracción de datos (representan recursos en forma abstracta) con variables que caracterizan el estado del recurso. Los monitores tienen como finalidad sincronizar y comunicar sistemas informáticos que cooperan haciendo uso de memoria compartida (12).

El programador, cuando hace uso de un monitor, solo se preocupa por el método y los argumentos que solicitan o retornan el recurso.

Si se ha comprobado el buen funcionamiento del monitor puede ser reusado.

Existen construcciones explícitas de monitores en distintos lenguajes de programación.

Los monitores son un importante mecanismo de control de concurrencia en la actualidad y en el futuro previsible (12).

Existen diferentes tipos de monitores que han sido publicados y evaluados

Andrews y Schneider (13) han clasificado los monitores, como con bloqueo o sin bloqueo de variables y han descripto técnicas de programación apropiadas para cada uno de ellos (basado en la clasificación de Howard).

Los componentes de un Monitor son: *el código de inicialización que establece el estado inicial de los recursos; las variables locales que almacenan el estado interno del recurso y el estado interno de algún procedimiento que son accedidas solamente desde el monitor; procedimientos internos que operan las variables locales; procedimientos que son exportados, los que son accedidos por procesos activos que acceden al monitor, estos procedimientos son los que nos permiten usar los recursos del monitor, y el control de la exclusión mutua que está basado en colas que se asocian al monitor, «colas del monitor»* (12).

**Gestión de las colas del Monitor:** cuando un proceso activo ejecuta uno de los procedimientos exportados del monitor, se dice que «Está Adentro del Monitor», el acceso al monitor garantiza que sólo un procedimiento este en el monitor; si hay un procedimiento en el monitor y otro trata de ingresar, este último se bloquea en la cola del monitor, y cuando un procedimiento abandona el monitor, selecciona el que está primero en la cola y lo desbloquea, si no hay ninguno el monitor queda libre.

**Condiciones de Sincronización de un Monitor:** si un proceso está en el monitor y no obtiene el recurso no puede seguir su ejecución, por lo que se requiere un mecanismo que lo bloquee hasta que el recurso esté disponible y pueda ser desbloqueado; este mecanismo es implementado con variables de condición, estas son accesibles solo desde el monitor, y con estas variables se realiza la gestión de sincronización (bloqueo y desbloqueo de los procesos en las colas del monitor).

**Operación Delay:** si  $c$  es una variable de condición,  $\text{delay}(c)$  hace que el proceso que la ejecuta se bloquee; antes de ejecutar  $\text{delay}$  por una condición, se debe desbloquear la cola de espera, de otra forma el monitor queda bloqueado, y por lo que  $\text{Delay}(c)$  debe liberar la exclusión mutua del monitor y bloquear el proceso que la ejecuto.

Según lo dicho un monitor puede tener tres colas: de entrada al monitor, de espera por las condiciones (pueden ser más de una) y para el proceso señalizador del desbloqueo.

### 3.3.4 Uso de redes de Petri en la construcción de monitores

Puede verse a un monitor como dividido en dos secciones, ellas son: la primera, referida a la política de colas que se debe ejecutar para lograr que sólo un proceso esté en el monitor, que se bloquen los procesos que no tienen los recursos y que se desbloquen los que obtuvieron los recursos, y la segunda comprende la lógica con que se administran los recursos.

Se plantea el problema del Productor/Consumidor como el representado por la PN de la Figura 8, la matriz de incidencia  $I$  y el vector de marcado inicial  $M_0$  correspondientes están ubicados debajo de la imagen. Más detalle del algoritmo Productor/Consumidor en la sección de Simulaciones y Mediciones (página 51).

Cada vez que el productor quiere insertar un elemento hay que disparar la transición  $T_0$ , para que el consumidor pueda sacarlo hay que disparar la transición  $T_1$ , una vez disparado  $T_0$  o  $T_1$  hay que dejar el sistema preparado, es decir que las transiciones  $T_0$  y  $T_1$  estén preparadas para ejecutar un disparo, con lo que hay que disparar  $T_3$  o  $T_2$  según sea  $T_0$  o  $T_1$  el disparo anterior. Esto último se puede realizar explícitamente o sistólicamente.

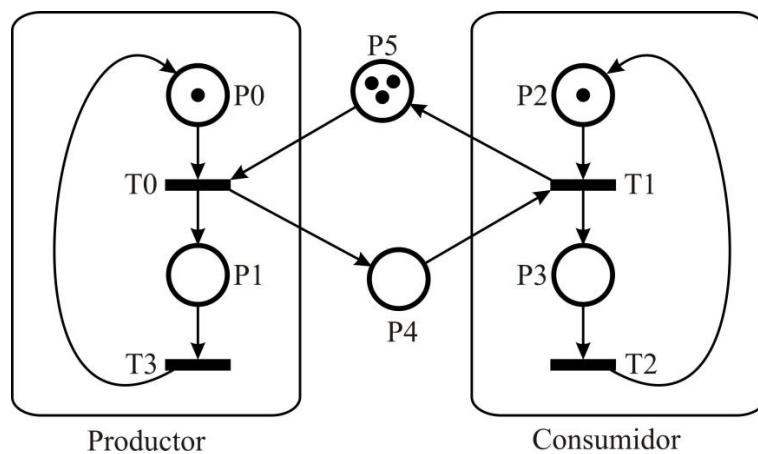


Figura 8. PN de un productor consumidor

Matriz de Incidencia I =

T0	T1	T2	T3	
P0	-1	0	0	1
P1	1	0	0	-1
P2	0	-1	1	0
P3	0	1	-1	0
P4	1	-1	0	0
P5	-1	1	0	0

Vector de Marcado Inicial  $M_0 =$

P0	P1	P2	P3	P4	P5
1	0	1	0	0	3

La PN realiza el trabajo de la lógica del monitor, es decir la exclusión mutua y/o la administración de recursos disponibles; esto es “cuando el vector de estado que resultó del disparo no tiene componentes negativas es porque el sistema está sincronizado o los recursos están utilizables, de otro modo el proceso debe ir a una cola hasta que el recurso o la sincronización esté disponible”.

Además debe remarcarse que el vector de estado indica si el disparo ha devuelto o tomado recursos, en el ejemplo cuando el valor en P5 (cantidad de token) disminuye es porque se ha insertado un elemento en el buffer y cuando aumenta es porque se ha sacado un elemento.

Ahora bien, como puede verse independientemente de las políticas implementadas en las colas del monitor la PN administra los recursos y la sincronización del sistema, puesto la PN es un modelo del sistema y lo que aquí se ha hecho es ejecutarla y evaluar el resultado con el fin de determinar si es posible la continuación de la ejecución del proceso o hilo que solicita el recurso o la sincronización.

La responsabilidad del algoritmo de Petri es calcular un estado después de un disparo para lo que realizar el siguiente cálculo.

$$m_i = m_0 + I \times \sum_{j=1}^i u_j = m_0 + I \times \bar{s}$$

Si el vector de disparo(s) solo realiza un disparo de una transición (u), todos sus componentes son cero excepto la del disparo que es uno, por lo cual no es necesario hacer la multiplicación con la matriz puesto que solo queda una columna, la del disparo, que se debe sumar con  $m_0$  para obtener el estado siguiente y luego evaluar que ninguna componente sea negativa para verificar que el estado es permitido o sea una transición permitida.

Cuando los disparos se realizan de uno a la vez podemos también tener calculado el éxito de los disparos como posibles para dar una respuesta inmediata a la solicitud de un recurso, esto es muy importante cuando se implementa por hardware puesto que mejora notablemente la latencia.

Para cambiar la lógica en esta solución solo es necesario cambiar la matriz de incidencia. Para implementar una política de prioridades en la solución solo hay que colocar las prioridades en el mismo orden que los **lugares** de la PN, y cuando se recorra la cola de los disparos no resueltos, que esperan en la cola de espera, si se lo hace por el índice del disparo también se está teniendo en cuenta la prioridad de los disparos; además pueden

ser implementadas otras prioridades y todo se reduce a la evaluación de la secuencia de disparos que están en la cola y los recursos disponibles, lo cual puede determinarse evaluando el vector de estado.

## 3.4 Sincronización y Redes de Petri

Las redes de Petri permiten describir de forma fácil y precisa sistemas en los que se producen relaciones de paralelismo, exclusión mutua y sincronización entre procesos que ocurren en ellos.

### 3.4.1 Exclusión mutua entre secciones Críticas

Cuando varios procesos que se ejecutan en paralelo comparten información, es necesario garantizar que los accesos a la información común no se lleven a cabo simultáneamente, para evitar problemas de interferencia que conduzcan a resultados erróneos.

Una solución a este problema es hacer mutuamente excluyentes las secciones críticas de los diferentes procesos encargados de acceder a la información. En la Figura 9, se muestra una PN que resuelve este problema, garantizando que los procesos acceden de a uno por vez a la sección crítica.

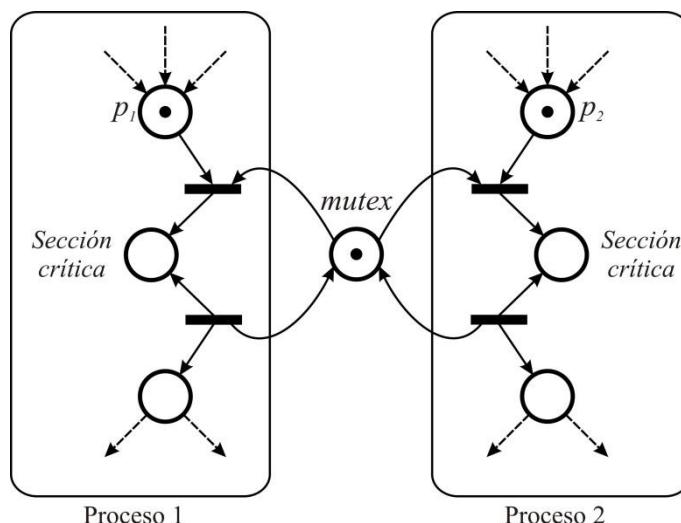
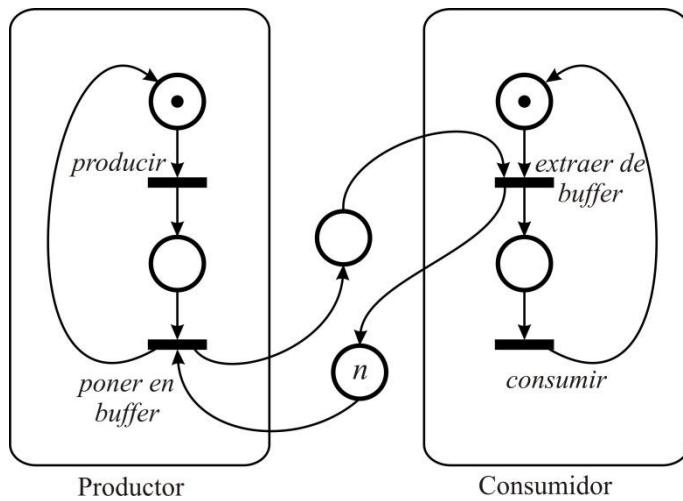


Figura 9. Red de Petri de una exclusión mutua

### 3.4.2 Sincronización de tareas

El problema *Productor/Consumidor* es uno de los ejemplos clásicos de acceso a recursos compartidos que debe arbitrarse mediante algún mecanismo de concurrencia que implemente la exclusión mutua. En este caso el proceso productor genera información que es colocada en un buffer de tamaño limitado y es extraída de este por un proceso consumidor. Si el buffer se encuentra lleno, el productor no puede colocar el dato, hasta que el consumidor retire elementos. Si el buffer se encuentra vacío, el consumidor no podrá retirar elementos hasta que el productor deposite en él. (Mayor detalle del Productor/Consumidor en la sección de Simulaciones y Mediciones, página 51).

Este problema involucra elementos de datos compartidos por ambos procesos y, en consecuencia, es necesario plantear tareas de sincronización entre ellos (procesos). En la Figura 10, se muestra un proceso de este tipo con un buffer limitado a “ $n$ ” unidades del producto.



**Figura 10. Redes de Petri de Productor/Consumidor con buffer limitado a “n” unidades.**



## 4 ESTUDIO PRELIMINAR

---

Como premisa para realizar un estudio preliminar, se plantea que la sincronización entre multiprocesadores múltiples hilos produce un aumento en la cantidad de instrucciones ejecutadas, generando así un incremento en el tiempo de ejecución de los procesadores. (1)

A través de las cuatro mediciones realizadas a continuación se estudia sobre un procesador de múltiples cores (real, no simulado) algunos parámetros de ejecución de: la escritura y lectura de una variable, un lock (semáforo binario) sobre una variable y la ejecución de un semáforo. Cabe desatascar que las primitivas de sincronización aquí estudiadas servirán como referencia para determinar el rendimiento del modulo a implementar.

Se establecen una serie de precondiciones que se van a cumplir a lo largo de todas las mediciones:

**Sistema Operativo:** Ubuntu Linux 10.10 64 bits

**Procesador:** Intel Core 2 Duo T6400

L1 Data = 32 KB (Asociativa 8 vías)

L1 Inst. = 32 KB (Asociativa 8 vías)

L2 = 2 MB (Asociativa 8 vías)

Line Size = 64 bytes

**Compilador:** gcc 3.4.6 (Ubuntu 3.4.6-1ubuntu2)

**Software Utilizado para hacer las mediciones:** **PCMTest** software escrito en C++, licencia GPL, y permite correr sobre Windows 2000 o superior y Linux de 32 o 64 bits.

El software utilizado ejecuta una cantidad configurable de veces el código que uno desea medir (código escrito en C++), y arroja valores acorde a las diferentes ejecuciones, estos son los siguientes: cantidad de ciclos de reloj por core que se emplearon para ejecutar ese código, la cantidad de Instrucciones ejecutadas y la cantidad de miss de cache producidos. Es importante mencionar que los datos obtenidos son similares en magnitud, pero no siempre serán iguales debido a la aleatoriedad de procesos ejecutados (por ejemplo, cantidad de procesos ejecutados del SO) en el procesador al realizar las mediciones.

La medición A1 verifica la cantidad de ciclos de reloj empleados para realizar la escritura de una variable sobre la cache de datos de primer nivel del procesador, similar al tiempo de escritura que consume el algoritmo del Escritor/Escritor (ver página 47) luego que ingresar a la zona de exclusión mutua.

<b>Medición A1</b>	
<b>Propósito</b>	Verificar la cantidad de ciclos de Clock empleados para llevar a cabo la escritura de una variable en Cache L1 de datos
<b>Dependencias</b>	
<b>Precondiciones</b>	Declarar e inicializar un vector tipo Integer.
<b>Acciones</b>	asignar un valor entero a una posición determinada del vector
<b>Resultados</b>	
<b>Esperados</b>	No deben producirse Miss de Cache
<b>Reales</b>	Cantidad de Instrucciones ejecutadas = 1 Ciclos de Reloj empleados = 44 Miss de Cache L1 = -                      Miss de Cache L2 = -

La medición A2 tiene como finalidad verificar la cantidad de ciclos de reloj empleados por el procesador para llevar a cabo la tarea de realizar un lock/unlock sobre una variable.

<b>Medición A2</b>	
<b>Propósito</b>	Verificar la cantidad de ciclos de Clock empleadas para ejecutar un Semáforo Binario
<b>Dependencias</b>	
<b>Precondiciones</b>	El Semáforo ya ha sido declarado e instanciado, siendo este ejecutado al menos una vez, antes de realizar el Test. No se toma en cuenta la cantidad de Clocks consumidos en declaración e inicialización de las variables empleadas
<b>Acciones</b>	Se va a ejecutar una instrucción Lock seguida de una Unlock sobre la misma variable
<b>Resultados</b>	
<b>Esperados</b>	No debe haber Miss de cache en L2. Debe haber Miss en L1.
<b>Reales</b>	Cantidad de Instrucciones ejecutadas = 47 Ciclos de Reloj empleados = 145 Miss de Cache L1 = 8                      Miss de Cache L2 = 0

La tercera medición (A3) es similar al anterior, pero en este caso entre el bloqueo y desbloqueo de la variable se realiza una lectura de un dato de tipo entero, este recrea el algoritmo del Productor/Consumidor (página 51) al momento de realizarse una extracción del buffer del proceso consumidor.

<b>Medición A3</b>	
<b>Propósito</b>	Verificar la cantidad de ciclos de Clock empleadas para ejecutar un semáforo Binario, y leer una variable del tipo integer.
<b>Dependencias</b>	
<b>Precondiciones</b>	El Semáforo ya ha sido declarado e instanciado, siendo este ejecutado al menos una vez, antes de realizar el Test. No se toman en cuenta los Clocks empleados en declaración, e inicialización de variables
<b>Acciones</b>	Se va a ejecutar una instrucción Lock, seguido se va a leer una posición específica de un vector del tipo Integer, luego se va a ejecutar la instrucción Unlock.
<b>Resultados</b>	
<b>Esperados</b>	No deben ocurrir MISS en Nivel L2
<b>Reales</b>	Cantidad de Instrucciones ejecutadas = 47 Ciclos de Reloj empleados = 145 Miss de Cache L1 = 2                      Miss de Cache L2 = 0

La medición A4 verifica los ciclos de reloj que son necesarios para la ejecución de un semáforo, siendo esto, la toma del semáforo por parte de un proceso, la lectura de una variable de tipo entera y la posterior liberación del semáforo tomado.

<b>Medición A4</b>	
<b>Propósito</b>	Verificar la cantidad de ciclos de Clock empleadas para ejecutar un semáforo, y leer una variable del tipo integer.
<b>Dependencias</b>	
<b>Precondiciones</b>	El semáforo se encuentra Inicializado, ya ha sido ejecutado al menos una vez. No se toman en cuenta los Clocks empleados en declaración, e inicialización de variables
<b>Acciones</b>	Se va a ejecutar una instrucción Wait, seguido se va a leer una variable del tipo Integer que se encuentra en cache, luego se va a ejecutar la instrucción Signal.
<b>Resultados</b>	
<b>Esperados</b>	La cantidad de Clocks debe ser superior a la Cantidad empleada en un acceso a memoria
<b>Reales</b>	Cantidad de Instrucciones ejecutadas = 27 Ciclos de Reloj empleados = 145 Miss de Cache L1 = 2                      Miss de Cache L2 = 0

La Tabla 2 resume para cada una de las mediciones anteriores los resultados obtenidos. Se agrega un dato adicional referido a la primera ejecución de cada uno de los algoritmos ejecutados, esa medición (primera vez) siempre es superior en magnitud a las siguientes debido a que los datos se encuentran almacenados en un nivel de memoria inferior (RAM).

	Ciclos Reloj	Inst. Ejecutadas	L1 Miss	L2 Miss
<b>ESCRITURA DE VARIABLE</b>	45	1	0	0
<b>(Primera Vez)</b>	100	3	0	0
<b>LECTURA DE VARIABLE DESDE MEMORIA</b>	45	2	1	0
<b>(Primera Vez)</b>	136	2	4	0
<b>LOCK/UNLOCK (con lectura de variable interior)</b>	145	47	2	0
<b>(Primera Vez)</b>	2991	1340	69	12
<b>SEMAFORO WAIT/SIGNAL (con lectura de variable interior)</b>	145	27	2	0
<b>(Primera Vez)</b>	3600	1515	57	8

Tabla 2. Resumen de las mediciones obtenidas.

## 4.1 Análisis de la Sincronización entre Hilos

En este análisis se logró conocer el tiempo que el procesador utiliza para llevar a cabo la sincronización entre hilos, como esta impacta en el rendimiento del mismo y la relación entre sincronizar y no sincronizar procesos. Este estudio se realiza empleando el algoritmo del Escritor/Escritor (pág. 47), utilizando este como modelo de test para la medición de la capacidad de sincronización de un sistema multithread, multicore. La sincronización entre los hilos de ejecución se lleva a cabo utilizando semáforos (14). El algoritmo realiza un número  $n$  de escrituras sobre un array compartido al ingresar a la sección crítica, de esta manera se logra variar  $n$  en cada ejecución para que junto con el numero iteraciones ( $I$ ) por hilo obtener un valor total de escrituras ( $E$ ). Cuando las  $n$  escrituras se realizan en exclusión mutua denominamos a este tiempo el de máxima sincronización ( $CS$ ). Por el contrario, si realizamos las  $E$  escrituras totalmente libres de sincronización lo denominamos tiempo sin sincronización ( $SS$ ).

La selección del algoritmo se debe a que es muy sencillo de aplicar y además la sobrecarga por sincronización es elevada (la cantidad de instrucciones utilizadas para sincronizar es mayor a la cantidad de instrucciones empleadas para ejecutar la tarea dentro de la exclusión mutua). Las mediciones se llevaron a cabo incrementando la cantidad

de elementos que se escriben (array compartido). Ejecutándose sobre el simulador SESC (Mayor detalle del código empleado en Anexo IV).

#### 4.1.1 Relación entre hilos sincronizados y sin sincronizar

Los resultados obtenidos durante el experimento se muestran en la Tabla 3. Claramente observamos que por más que se incremente el número de escrituras totales realizadas, para igual numero de escrituras  $n$  en exclusión mutua, la relación porcentual para los tiempos de ejecución permanece constante.

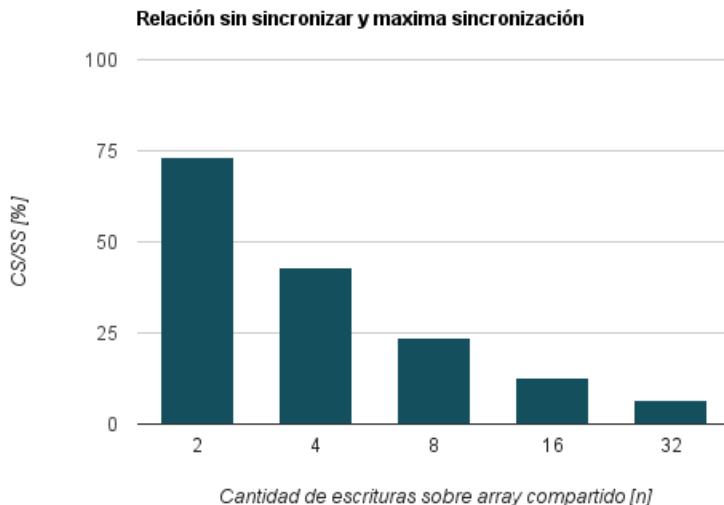
La relación porcentual entre la máxima sincronización, en la cual cada hilo escribe solo  $n$  veces por sincronización realizada, y la misma cantidad de escrituras por hilo no sincronizadas es arrojada en la Figura 11. En ella se observa claramente como la relación decrementa a medida que  $n$  aumenta, esto se debe a que la cantidad de escrituras aumentan para modificar el array compartido en cada iteración respecto de la cantidad de semáforos utilizados para sincronizar, los cuales permanecen constantes. Ocasionando esto que para valores grandes de  $n$  los tiempos consumidos para sincronizar los procesos se tornan despreciables en relación a las escrituras realizadas.

Iteraciones	Cantidad de escrituras sobre array en exclusión mutua ( $n$ )	Tiempo sin sincronizar (ms)	Tiempo con sincronización máxima (ms)	Relación CS/SS	%
5000	2	0.759	1.313	1,72990777	73
10000	2	1.517	2.626	1,73104812	
20000	2	3.034	5.251	1,73071852	
50000	2	7.584	13.126	1,73074895	
5000	4	1.284	1.838	1,43146417	43
10000	4	2.567	3.676	1,43202182	
20000	4	5.134	7.351	1,43182704	
50000	4	12.834	18.376	1,43182172	
5000	8	2.334	2.888	1,23736075	24
10000	8	4.667	5.776	1,23762588	
20000	8	9.334	11.551	1,23751875	
50000	8	23.334	28.876	1,2375075	
5000	16	4.434	4.988	1,12494362	13
10000	16	8.867	9.976	1,12507049	
20000	16	17.734	19.951	1,12501411	
50000	16	44.334	49.876	1,12500564	
5000	32	8.634	9.188	1,06416493	6
10000	32	17.267	18.376	1,06422656	
20000	32	34.534	36.751	1,0641976	
50000	32	86.334	91.876	1,06419255	

Tabla 3. Relación sincronización vs. no sincronización

Nota: *CS* indica tiempo con sincronización, *SS* indica tiempo sin sincronización.

La siguiente figura grafica la relación porcentual de la columna derecha de la tabla anterior.



**Figura 11. Algoritmo Escritor/Escritor (máxima sincronización vs. sin sincronización)**

Esto indica que los métodos de sincronización que estamos utilizando van a tener una gran incidencia sobre hilos que realizan gran cantidad de modificaciones alternadas de una variable compartida.

#### 4.1.2 Relación entre la cantidad de escrituras por sincronización respecto de hilos no sincronizados

En este experimento se utilizó nuevamente el algoritmo Escritor/Escritor pero esta vez se mantuvo constante la cantidad de escrituras totales a medida que se incrementaba la cantidad  $n$  de escrituras. Se realizaron como para el caso anterior mediciones sincronizando los hilos que escriben sobre el array compartido, como sin sincronizar los mismos.

La Tabla 4 y la Figura 12 muestran los resultados obtenidos. *La principal observación que se realiza a partir de la tabla es que al incrementar las escrituras realizadas en la región crítica ( $n$ ) disminuye la diferencia de tiempos de ejecución entre el algoritmo sincronizado respecto al sin sincronizar. Concluyendo que la aplicación de una mejora sobre las primitivas de sincronización tiene el mayor impacto sobre algoritmos en los que el tiempo empleado para sincronizar sea similar al empleado en la región crítica. A este tipo de algoritmo lo denominaremos fuertemente sincronizados.*

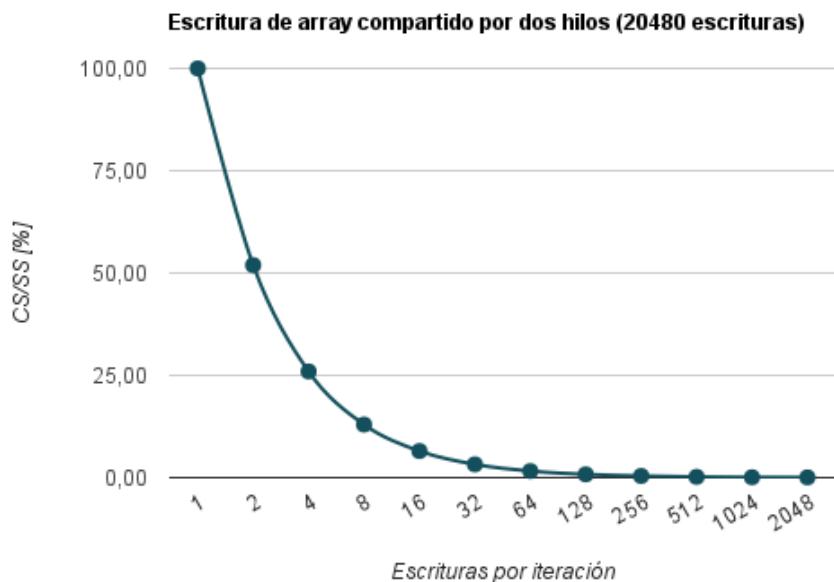
Escrituras por iteración (n)	Tiempo sin sincronizar (ms)	Tiempo sincronizando (ms)	Relación porcentual CS/SS
1	35.363	38.469	<b>100.00</b>
2	35.363	36.976	<b>51.93</b>
4	35.363	36.169	<b>25.95</b>
8	35.363	35.766	<b>12.97</b>
16	35.363	35.565	<b>6.50</b>
32	35.363	35.464	<b>3.25</b>
64	35.363	35.413	<b>1.61</b>
128	35.363	35.388	<b>0.80</b>
256	35.363	35.376	<b>0.42</b>
512	35.363	35.369	<b>0.19</b>

Escripturas por iteración (n)	Tiempo sin sincronizar (ms)	Tiempo sincronizando (ms)	Relación porcentual CS/SS
1024	35.363	35.366	0.10
2048	35.363	35.365	0.06

**Tabla 4.** Relación sincronizar vs. no sincronizar

Nota: *CS* indica tiempo con sincronización, *SS* indica tiempo sin sincronización. Se toma el 100% como la relación porcentual para una sola escritura sobre el array compartido en la región crítica. Los demás valores son relativos a este.

En la Tabla 4 no se indica la cantidad de iteraciones totales, ya que vimos en la Tabla 3 que la relación entre los tiempos permanece constante independientemente de la cantidad de iteraciones realizadas.

**Figura 12.** Relación porcentual sincronizar vs. no sincronizar.

Nota: Se toma el 100% como la relación porcentual para una sola escritura sobre el array compartido en la región crítica. Los demás valores son relativos a este.

Escripturas por sincronización	Iteraciones	Cantidad de Instrucciones (sincronizando)	Ciclos de Reloj (sincronizando)
1	500000	28262376	94823463
2	250000	15974426	54887740
4	125000	11110365	39476464
8	62500	8959924	32820546
16	31259	7910174	29492509
32	15625	7321114	27591746
64	7813	7026174	26641346
128	3906	6893884	26225346
256	1954	6826104	26017346
512	977	6788944	25913346
1024	488	6763227	25859947

**Tabla 5.** Instrucciones y ciclos de reloj con algoritmo sincronizado.

En la Tabla 5 se muestran la cantidad de instrucciones ejecutadas y ciclos de reloj consumidos por el procesador para el algoritmo donde las escrituras son sincronizadas. *Se ve claramente que al aumentar el número de iteraciones, por ende estamos disminuyendo la cantidad de sincronizaciones realizadas, disminuyen la cantidad de instrucciones ejecutadas y los ciclos de reloj consumidos.*

*De los experimentos realizados podemos concluir que la sincronización para procesos a los que denominamos **fuertemente sincronizados** tiene un incremento superior al 25% en el tiempo de ejecución, siendo este campo factible para el análisis de mejoras.*

## 4.2 Análisis de las alternativas de sincronización por hardware

*Teniendo en cuenta los Objetivos y Restricciones planteados, como así también los resultados obtenidos en el estudio realizado sobre el impacto de la Sincronización, se van a analizar distintas formas de implementar mecanismos de sincronización, y seleccionar el mejor, para llevar a cabo la sincronización entre hilos corriendo sobre diferentes procesadores a nivel de hardware. Con el fin de decrementar los tiempos de ejecución de procesos fuertemente sincronizados.*

### 4.2.1.1 Alternativas de implementación

Las alternativas que fueron contempladas para llevar a cabo la implementación del módulo de hardware capaz de realizar la sincronización de procesos concurrentes son:

- Módulo incorporado al procesador controlado por nuevas instrucciones.
- Módulo independiente actuando como coprocesador para gestionar un sistema de colas (7)
- Módulo a nivel de RAM comunicado a través de posiciones de memoria compartida.
- Módulo a nivel de Cache comunicado a través de posiciones de memoria compartida.

#### 4.2.1.1.1 Breve análisis de las opciones

##### 4.2.1.1.1.1 Módulo incorporado al procesador controlado por nuevas instrucciones

Crear una nueva unidad funcional dentro del procesador que permita la sincronización entre los distintos procesadores, comunicados a través de nuevas instrucciones. Implica que se debe modificar el Set de Instrucciones (ISA) de cada procesador, debiendo así modificar la arquitectura de cada procesador sobre el cual se desee implementar, haciendo esta posibilidad inviable, ya que no respeta las restricciones planteadas con anterioridad.

##### 4.2.1.1.1.2 Módulo independiente actuando como coprocesador para gestionar un sistema de colas

Esta opción utiliza la primitiva de monitor mediante el algoritmo de Redes de Petri. El módulo funciona como coprocesador el cual ejecuta un monitor para hacer la gestión de un sistema de colas y así sincronizar los diferentes procesos que corren en cada procesa-

dor. Esta opción es inviable ya que no cumple con la restricción de ser instanciable en cualquier procesador.

#### **4.2.1.1.3 Módulo a nivel de RAM comunicado a través de posiciones de memoria compartida**

Crear un módulo que implemente la sincronización a nivel de RAM el cual se comunique mediante direcciones de memoria compartida con los distintos procesadores, con el fin de sincronizar los procesos. Esta opción cumple con las condiciones necesarias, con la salvedad de que la comunicación con la RAM tendrá una mayor latencia respecto a los niveles más altos en la jerarquía de memoria (cache L1, l2).

#### **4.2.1.1.4 Módulo a nivel de Cache comunicado a través de posiciones de memoria compartida**

La idea que persigue esta, es utilizar una memoria cache, la cual implemente un controlador que permita llevar a cabo la sincronización de los procesos, empleando la teoría de redes de Petri. Esta propuesta es la mejor, y más factible de implementar ya que respeta los objetivos y las restricciones planteadas con anterioridad. Además posee una mayor velocidad de ejecución respecto a niveles más bajos en la jerarquía de memoria, como así también permite ser implementada en SESC, utilizando módulos ya funcionando y con la menor reingeniería sobre el mismo.

A continuación se presenta una tabla donde se comparan las distintas opciones que se listaron anteriormente para implementar los mecanismos de sincronización, según las restricciones planteadas en los objetivos del proyecto.

	<i>Módulo con nuevas instrucciones</i>	<i>Coprocesador con sistema de colas</i>	<i>Memoria compartida RAM</i>	<i>Memoria compartida Cache</i>
<i>No modificar ISA</i>	No cumple	No cumple	Si cumple	Si cumple
<i>No requerir operaciones del SO</i>	No cumple	Si cumple	Si cumple	Si cumple
<i>Cambios a nivel de programa mínimos</i>	No cumple	Si cumple	Si cumple	Si cumple
<i>Cambios en hardware con módulos Testeados</i>	No cumple	No cumple	Si cumple	Si cumple
<i>Viabilidad en SESC</i>	Si cumple	No cumple	Si cumple	Si cumple
<i>Velocidad de ejecución</i>	Muy alta	Alta	Muy baja	Alta

**Tabla 6. Alternativas de implementación vs. Restricciones planteadas**

A partir de analizar y estudiar las distintas posibilidades de implementación, se decidió que la opción para llevar a cabo el desarrollo es: **Módulo a nivel de Cache comunicado a través de posiciones de memoria compartida**. Esta elección se debe a que cumple con todas las restricciones planteadas. Y a su vez es la alternativa de mayor velocidad de ejecución, obteniendo así un mejor rendimiento del sistema.



## 5 HIPÓTESIS

---

Como hipótesis se plantea la simulación de un módulo que implemente el algoritmo de redes de Petri a nivel de cache L1, comunicando a través de posiciones de memoria compartida con el fin de disminuir los tiempos actuales de sincronización por semáforos, dichos semáforos están implementados por el sistema operativo.

### 5.1 Resultados Esperados

Se espera que una vez implementada la mejora (módulo) en el simulador, los tiempos de ejecución de algoritmos fuertemente sincronizados y la cantidad de instrucciones ejecutadas a nivel de procesador disminuyan de un 20% a un 30% respecto a los mismos valores (tiempo, cantidad de instrucciones) obtenidos con los procesadores estándares usados en el simulador. Se tomará el procesador openSparc T1 para realizar las simulaciones sobre SESC original, y también se le agregará el módulo de sincronización de Petri (SESC modificado) para efectuar las simulaciones de manera análoga al SESC original.



## 6 DESARROLLO E IMPLEMENTACIÓN

---

En esta sección se hace una breve reseña del simulador y cuales son las partes que son modificadas, se plantea el algoritmo de Petri a implementar dentro del módulo y cómo se logra incorporar al simulador. También se explica la nueva arquitectura y el funcionamiento de la misma. Una vez analizada toda la implementación se realizan cuatro mediciones para determinar la performance de la nueva arquitectura.

### 6.1 Introducción a SESC

En primera instancia, es importante remarcar el hecho de que el simulador SESC inicia como un proyecto “mascota” de José Renau mientras realizaba un PhD en Urbana-Campaing en el grupo IACOMA en el año 2003; por este motivo, dicho simulador, no es de tan amplio conocimiento y no posee una extensa documentación, lo que ocasiona que el trabajo de análisis sea más laborioso. Como contrapartida la flexibilidad en la configuración de las características de los procesadores simulados y los niveles de memoria instantiados genera un alto beneficio a la hora de incorporar el módulo de sincronización de Petri.

Se menciona brevemente a continuación el estudio del Simulador que se llevó a cabo para poder aplicar el Mecanismo de Sincronización anteriormente seleccionado.

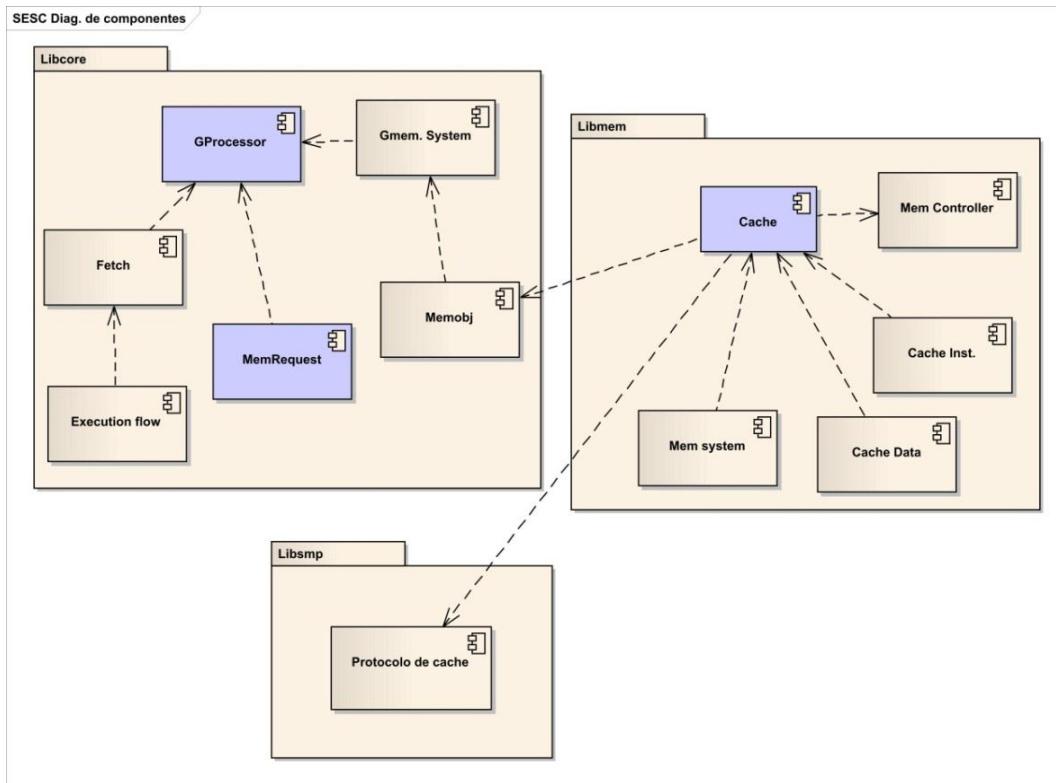
El simulador se encuentra separado en paquetes y cada uno de estos contiene las diferentes clases que realizan las tareas pertinentes a cada parte de un procesador. En la Figura 13 se pueden apreciar los paquetes más importantes con las clases de mayor interés, siendo estas las que se estudiaron con más detalle para lograr la implementación.

**Libcore** en conjunto con **Libmem** son los paquetes de clases más importantes puesto que contienen todo lo relacionado a la memoria del procesador simulado (procesador, objetos de memoria como cache L1, L2).

Dentro de estos paquetes se encuentran las clases que se encargan de: la creación de las memorias y de los buses mediante lectura de parámetros del archivo de configuración, funcionamiento de la memoria, controlador de cache, protocolo de remplazo de datos, protocolos de coherencia de cache, accesos a las memorias, etc.

Habiendo realizado un estudio exhaustivo de estas clases, se logra comprender y reconocer el funcionamiento general del procesador simulado. Por cuestiones de practicidad se considera que no es necesario dar detalle de estas y de sus composiciones en este documento, encontrándose en el documento electrónico (DVD) adjunto, dicha documentación.

Luego de comprender el funcionamiento del simulador y seleccionar el mecanismo encargado de llevar a cabo la sincronización, se está en condiciones de detallar el Algoritmo a implementar en el mecanismo.



**Figura 13. Diagrama de componentes simplificado de SESC**

*Nota: Los componentes coloreados son aquellos que fueron modificados para incorporar el módulo a implementar.*

## 6.2 Algoritmo de Petri

*Este algoritmo permite la ejecución de la Red de Petri, es decir la evolución de los estados a través de los disparos de las transiciones.*

**Restricción:** Los disparos se ejecutan de a uno por vez, es decir el vector de disparo contiene solamente un elemento igual a uno y el resto son ceros. Y una transición puede tener solo un disparo pendiente de ejecución. Estas restricciones no implican una disminución en el poder de expresividad de las redes. (3)

A partir de la expresión  $M_{j+1} = M_j + I \times \sigma$  donde  $I$  es la Matriz de Incidencia,  $M_j$  el vector de marcado después del disparo  $j$ ,  $\sigma$  el vector disparo. Siendo la matriz  $I$  formada por las columnas  $I^1, I^2, \dots, I^n$  con las cuales se obtienen los vectores  $S^i$ , efectuando las operaciones que indica la siguiente formula:  $S^i = (M_0 + I^i)$

A partir de cada  $S^i$ , definimos el escalar  $E_i$ , según la siguiente condición: Si  $S^i$  contiene algún elemento menor a cero es  $E_i = 0$  de lo contrario es  $E_i = 1$ .

Los cálculos de todas las variables  $E_i$  se obtienen de forma paralela según la cantidad de columnas  $I^1, I^2, \dots, I^n$

Siendo el vector de disparo  $\sigma_i$ , el cual tiene el elemento  $i$ -ésimo igual a uno y el resto cero.

Los pasos del algoritmo al recibir un disparo son:

1. Se evalúa que el disparo cumpla con las restricciones planteadas. Si es correcto se pasa al punto 2. De lo contrario el disparo es eliminado.
2. Se evalúa una transición en la red caracterizada por el vector de disparo  $\sigma_i$ , cuyo  $i$ -ésimo elemento se utiliza para seleccionar  $E_i$ , con el fin de ejecutar o no la transición dependiendo de si  $E_i$  vale uno o cero respectivamente.
  - a. Si la transición no puede realizarse debido a  $E_i = 0$  el disparo es puesto en una cola de disparos pendientes y se va al punto 1.
  - b. Si la transición puede ejecutarse correctamente se remplaza  $M_j$  por  $S^i$ , se calculan todos los  $S^i$ .
3. Luego se controla toda la cola de disparos pendientes para verificar si es posible el disparo de alguno de estos, ejecutándose en caso de ser posible. Si se produce un disparo exitoso se repite el paso 3 hasta que no exista ninguno posible.
4. Se queda a la espera de un nuevo disparo.

El objetivo que persigue este algoritmo es el de tener todos los escalares ( $E_i$ ) calculados a la hora de efectuarse el disparo, para obtener el menor tiempo de respuesta, ya que está pensado para ser implementado en hardware.

### 6.3 Algoritmo de Petri Implementado

Se crea un archivo el cual contiene la matriz de incidencia  $I$ , el vector de marcado inicial  $M_0$ , la cantidad de vectores disparo y cada uno de estos ( $\sigma$ ). Este archivo se utiliza como inicialización del programa que se desea ejecutar (Escritor/Escritor, Productor/Consumidor, etc.). Cada programa a ejecutar posee su propio archivo de inicialización. Cuando el programa necesita acceder a alguna variable en exclusión mutua (se debe sincronizar su uso), se efectúa el disparo sobre una variable (posición de memoria asociada al disparo), este “disparo” (petición de escritura de un dato) indica a qué variable sincronizada se está queriendo acceder. **“Dicho disparo es reconocido por el algoritmo de Petri que ejecuta la cache, actuando como controlador de cache”** y se encarga de efectuar las operaciones pertinentes para obtener el nuevo estado de la red (nueva marcación). Según el resultado de estas operaciones matemáticas, se determina si es posible acceder o no a la variable deseada (sincronización). Debe aclararse que este mecanismo que se encuentra en la memoria cache de Petri, solo almacena las variables utilizadas para llevar a cabo la sincronización, el dato que se desea modificar (variable en exclusión mutua) se encuentra en L1 y/o L2 y/o en RAM. En el caso que no se pueda acceder a la variable a sincronizar, dicho disparo es almacenado en una cola de disparos pendientes, la cual es controlada cada vez que se efectúa un disparo de forma exitosa.

Si el disparo es exitoso la cache devuelve un *Hit* al procesador y la ejecución de proceso continua. En cambio, si el disparo no es posible se retorna un *Miss*, deteniendo la ejecución del proceso hasta que el disparo sea posible, lo cual es comunicado con un *Hit*.

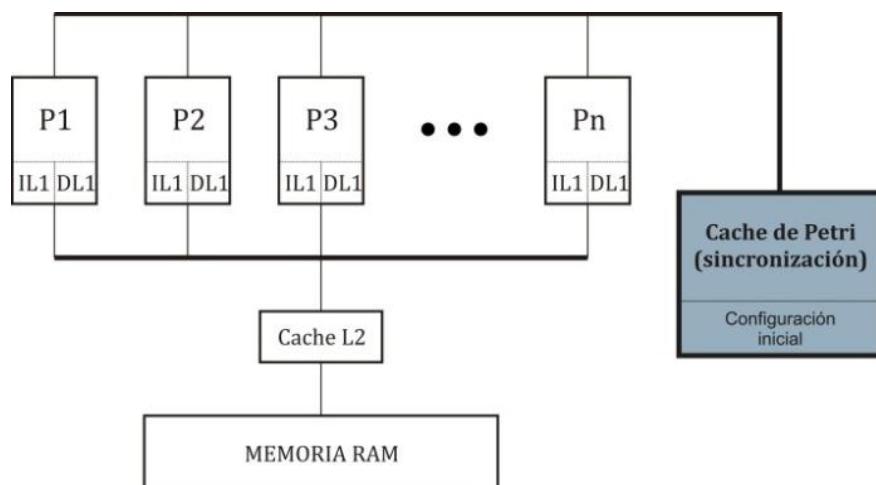
El mecanismo aquí expuesto permite realizar la sincronización en los tiempos requeridos para una escritura en cache y una comparación en un registro de cache, lo que implica el

menor tiempo posible. Y además, cuando una sincronización pendiente es resuelta el tiempo de comunicación es el del envío del *Hit*.

### 6.3.1 Implementación del Mecanismo en el Simulador

En puntos anteriores se ha explicado el algoritmo para llevar a cabo la implementación, ahora bien se explicará con más detalle a nivel de implementación y ejecución en el simulador.

Para poder llevar a cabo la opción “**Módulo a nivel de Cache comunicado a través de posiciones de memoria compartida**”. Primero es necesario que la Cache sea capaz de implementar un mecanismo de inicialización el cual le permita definir: el marcado inicial, la matriz de incidencia y los vectores de disparos posibles. Una vez cargado el estado inicial, los disparos de transiciones retornarán el éxito o fracaso del mismo, encolando los disparos que no fueron ejecutados para testearlos luego de un disparo exitoso. Todo este mecanismo de Petri esta implementado dentro de una cache instanciada en un nivel 1 ya que va a estar directamente conectada a todos los procesadores, pero va a ser compartida por todos como una cache de nivel 2. El algoritmo de Petri (procesador de Petri) se encarga de hacer de “controlador de cache”. Se encuentra en Nivel L1 de cache para lograr un rápido acceso, (más rápido que L2), y no se encuentra dentro del procesador ya que esta cache almacena el estado de la red de todo el sistema de procesadores, por lo tanto debe ser compartida por todos los procesadores como una cache de Nivel 2, es por esto que debe ser un nuevo tipo de memoria cache. No se encuentra conectado a RAM, debido a que no necesita de esta para poder funcionar ya que los datos que se encuentran en esta cache no necesitan ser remplazados durante la ejecución del programa. En la Figura 14 se muestra el nivel de memoria cache a incorporar.



**Figura 14. Memoria Cache a Implementar**

Aprovechando el modo jerárquico que posee el simulador para instanciar la memoria desde un archivo de configuración, podemos crear la petriCache la cual no se encuentra asociada a ningún procesador particular, sino que se conectada directamente a todos los procesadores como lo muestra la Figura 14. Para poder compartir la memoria de esta manera hubo que realizar modificaciones dentro de la clase *GMemorySystem* de SESC que es la encargada de instanciar todos los objetos del tipo memoria, en esta se incluyó una nueva variable denominada *petriSource* la cual toma como parámetros los valores definidos en el archivo de configuración del simulador, dentro de este archivo donde se encuentra la etiqueta que denota la configuración del procesador (se encuentra en este apartado porque el módulo de Petri debe ir conectado a todos los procesadores) declara-

mos *petriSource* = "PMemory PL1 shared" definiendo al módulo como una memoria instanciada en el primer nivel, compartida todos los procesadores y que va a ejecutar una red de Petri como mecanismo de sincronización entre procesos. La memoria se la etiquetó como *PMemory*, como vimos anteriormente, así que definimos esta nueva sección (poner la etiqueta entre corchetes para declarar una sección) del archivo de configuración del simulador y se lleva a cabo la configuración de parámetros como *tipo de cache, tamaño, algoritmo de Petri, nivel inferior de memoria*, etc. (ver Anexo II para tener una mejor perspectiva de las configuraciones enunciadas).

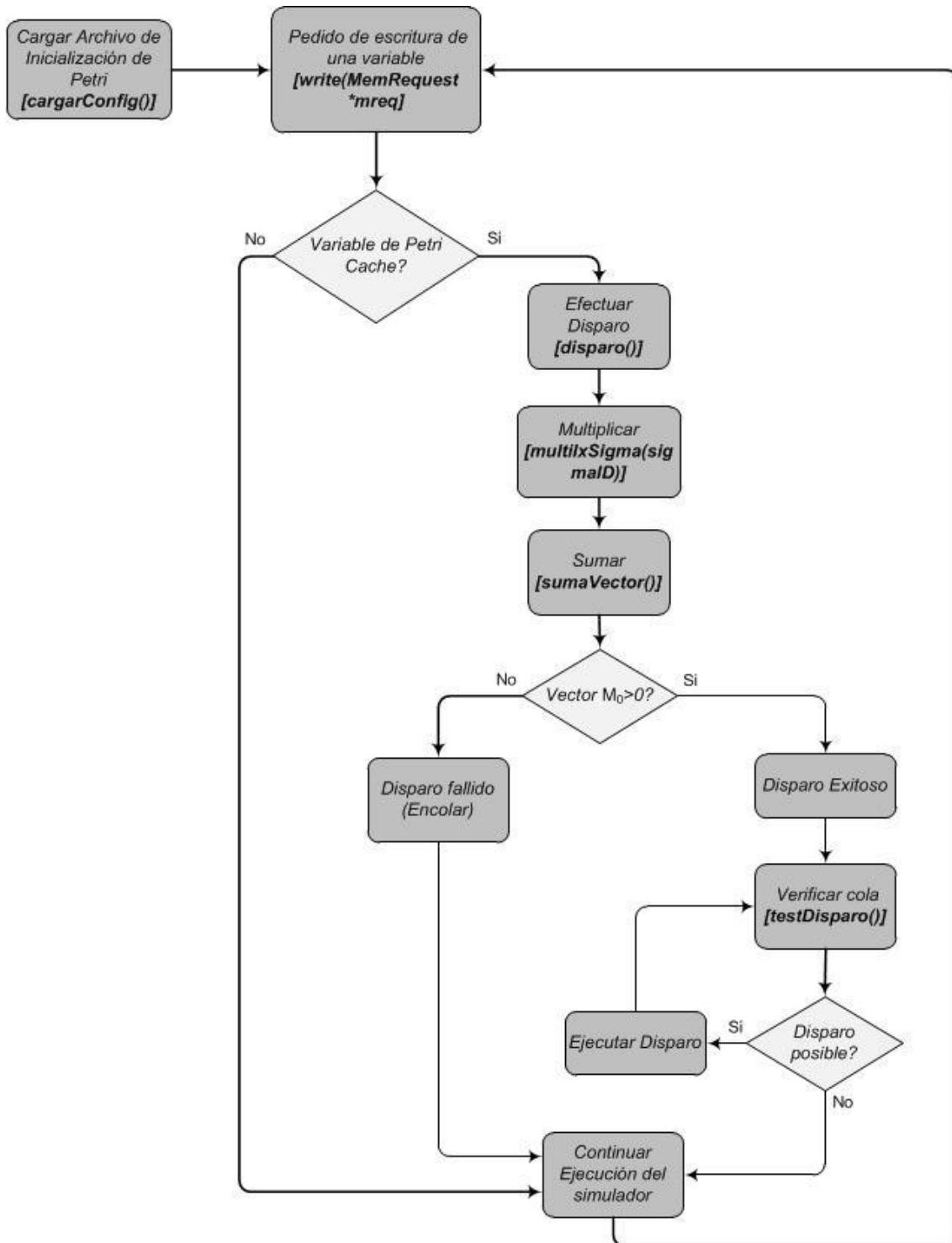
En el nivel de memoria inferior como vemos en la Figura 14 no existe ningún bus que conecte a la Petri Cache con la RAM, seteándose este parámetro a la variable *lowerLevel* de la memoria de Petri.

De la clase Cache hereda la nueva clase declarada como PETRICache, la cual implementa el algoritmo de Petri (explicado anteriormente) en distintos métodos, los que detallaremos a continuación. (Para mayor detalle ver el código del simulador en el documento electrónico (DVD)).

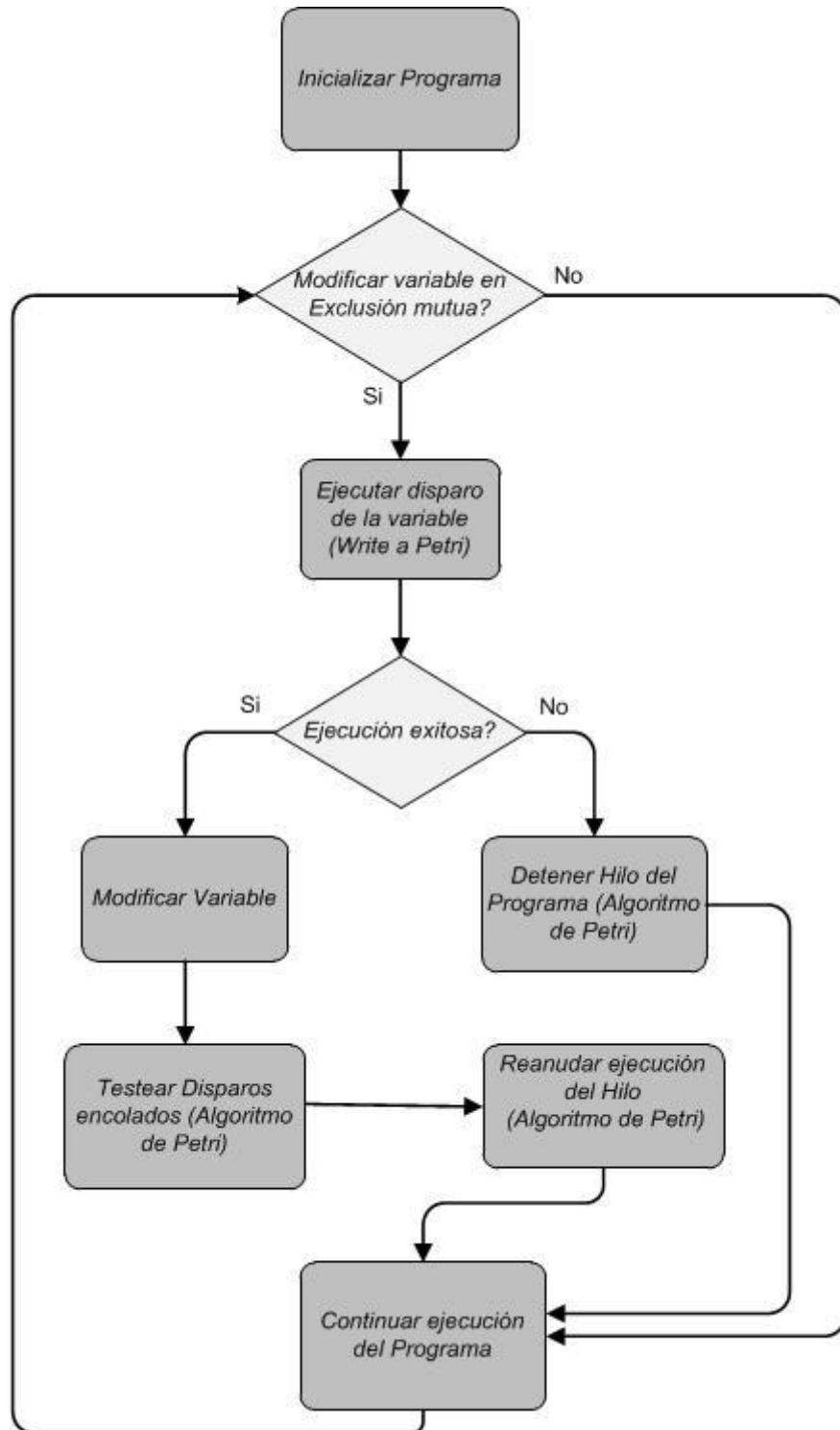
El método void PETRICache::cargarConfig() se encarga del mecanismo de inicialización mediante la carga del archivo de configuración del algoritmo de petri, este contiene: id, cantidad de filas, de columnas, matriz de incidencia, marcado inicial de la red, cantidad de vectores disparos, vectores disparo. Una vez ya configurado e inicializado el algoritmo de petri (controlador de cache), se continúa detallando los aspectos funcionales del algoritmo de Petri. En la clase *MemRequest* se encuentra la sección del código que controla si la petición de memoria es de dato o instrucción correspondiente a Nivel 1 de cache, o si se está pidiendo un dato de la cache de Petri (dato manejado por el controlador de cache). Según si se pide uno u otro se ejecutan o no aquellos métodos del controlador de cache. Suponiendo que se haga una petición de un dato sincronizado por la cache de Petri, en otras palabras luego de que se efectúe un disparo (el programa que se ejecuta quiere escribir un dato en una dirección de memoria manejada por nuestro controlador de cache de Petri) se llama al método PETRICache::write(*MemRequest* \**mreq*), el cual toma un lock para ejecutar con exclusión mutua el método PETRICache::disparo(), este llama al método que multiplica la matriz I por el vector Sigma, multiIxSigma(*sigmaID*), y con ese resultado se llama al método sumaVector(), el cual suma el vector M0. Una vez efectuada la operación ( $M = M0 + I * \Sigma$ ) se procede a controlar el vector M, si este contiene algún número negativo no se puede efectuar el disparo y se deja constancia del disparo fallido (es encolado en una FIFO para verificarlo posteriormente), en el caso de que se pueda efectuar (todos los valores positivos) se deja constancia de disparo efectivo o realizado, marcándose como *Hit* (permite verificar en el log que la cantidad de disparos realizados es coherente con la cantidad de disparos que ejecutó nuestro programa). Luego de cada disparo realizado se llama al método testdisparo(), el cual se encarga de testear los disparos encolados para una nueva verificación debido a que por el disparo posible cambió el marcado de la red, esto se realiza hasta que al recorrer la cola todos los disparos no sean posibles, también se cuentan los *Miss* producidos por disparos fallidos.

Al finalizar los disparos, se procede a liberar el lock adquirido.

En la Figura 15 se muestra un diagrama de flujo que detalla de forma sencilla el funcionamiento del algoritmo implementado dentro del simulador, resaltando como se realiza el llamado de los distintos métodos entre sí, los cuales están descriptos con anterioridad.

**Figura 15. Diagrama de flujo del funcionamiento del algoritmo de Petri implementado.**

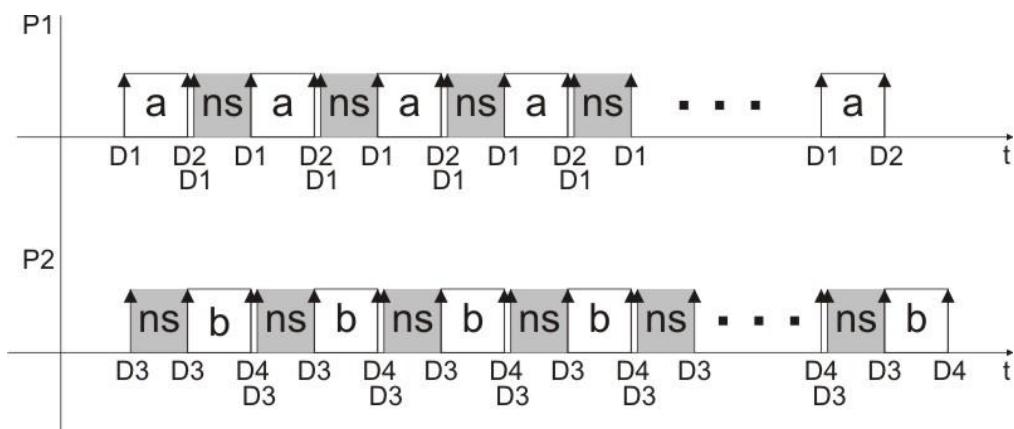
A modo de brindar mayor detalle se muestra la Figura 16, la cual muestra el diagrama de flujo básico que cumplen todos los programas que se vayan a ejecutar sobre SESC, en dicha figura se puede apreciar como es el formato de estos programas y como se vincula con el algoritmo de Petri en el simulador. (Mayor detalle en el Anexo, donde se presentan los códigos empleados en las mediciones)

**Figura 16. Diagrama de flujo básico de la ejecución de un programa.**

A la hora de desarrollar la implementación en el simulador, se encontraron una serie de problemas, uno de estos fue que el simulador esta hecho de tal forma que la ejecución del programa se hace sobre un emulador (SESC posee un emulador que ejecuta las instrucciones, descripto en la página 71), mientras que la contabilización de los tiempos, accesos, fallos, aciertos, etc., se realiza a nivel de simulación. La cache de Petri, solo fue posible de instanciar a nivel de Simulación, teniendo de esta forma una serie de inconvenientes. No se pueden manipular correctamente los hilos que llevaban a cabo la ejecución

de las instrucciones del programa, ya que no se tienen las instrucciones en la API del simulador para hacer el correcto tratamiento de los mismos, ocasionando así otro problema. Si el disparo no se puede realizar, SESC no es capaz de frenar el hilo correspondiente a nivel de emulación, debido a la forma en que se encuentra creado el simulador, se computan de forma correcta los Miss correspondientes a no poder efectuar el disparo, pero igual sigue corriendo el programa cuando debería quedarse “parado”, hasta que otro hilo realice el disparo capaz de modificar el marcado de matriz y ahí poder efectuar el disparo del hilo “parado”, continuando con la ejecución.

Entonces, la secuencia de disparos, por ejemplo para una exclusión mutua como la presentada en la Figura 9, que en teoría debiera respetar el orden que se muestra en la Figura 17, no se cumple, ya que los hilos de ejecución no se “paran” cuando no se puede ejecutar el disparo.



**Figura 17. Ejecución de disparos por 2 procesos P1 y P2. El proceso P1 efectúa el disparo D1, cuando el proceso P2 está ejecutando el disparo D3, el proceso P1 se encuentra en ns (No Shot) ya que los subprocesos a y b no se pueden dar de forma simultánea, deben estar sincronizados.**

Debido a este problema se debe buscar una solución. Luego de un profundo estudio y análisis, se encuentran dos posibles soluciones:

1. **Efectuar una espera activa** por parte de los hilos hasta que se cumplan las condiciones para efectuar el disparo.
2. **Emplear semáforos** (ya que estos si son provistos en la API del simulador) para forzar la detención de los hilos cuando el disparo no es posible, y que el semáforo sea liberado por aquellos hilos que efectúan correctamente los disparos.

La primera solución tiene el problema de que se necesita soporte para hilos por parte de la API, lo cual no es posible, por lo cual se descarta, por ende la única opción con la que se cuenta para una correcta ejecución, es la de emplear semáforos, con el fin de poder respetar la teoría en la Figura 17.

Mediante el empleo de instrucciones especiales provistas por la API del simulador se ignora el tiempo empleado en ejecutar los semáforos, (no es tenido en cuenta a la hora de computar los tiempos de ejecución), ya que estos semáforos son una “medida para lograr el cometido de la simulación”, sujeto a como está programado el simulador.

Como podemos ver la solución seleccionada está en completa concordancia con el objetivo de esta simulación.

## 6.4 Simulaciones Y Mediciones

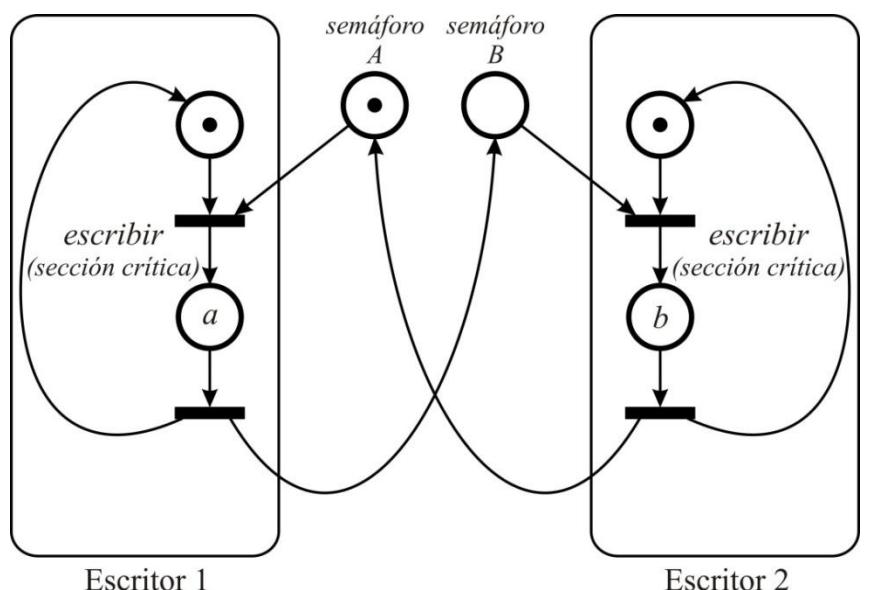
Para la simulación y medición de los tiempos de ejecución se emplean algoritmos que requieran de accesos simultáneos a recursos compartidos, los cuales sean fuertemente sincronizados y que puedan ser expresados mediante un grafo de Red de Petri, estos son:

1. *Escrivtor / Escritor*
2. *Productor / Consumidor*
3. *Algoritmo de Simulación de una Planta de Embalajes*
4. *Algoritmo de control de un mecanismo de velocidad crucero con detección de automóviles.*

### 6.4.1 Escritor / Escritor

Este es el caso en que dos procesos tratan de escribir en el mismo buffer, accediendo de forma alternada.

El hilo principal (el que corre sobre el procesador 0) instancia dos hilos (los cuales se ejecutan sobre los procesadores 1 y 2 respectivamente). Este par de hilos realizan las escrituras (*ambos escriben*) sobre un array compartido de datos, las cantidad n de escrituras no va a ser fija, y se harán varias simulaciones para obtener así resultados coherentes. La Figura 18, representa el modelo del Escritor/ Escritor por una Red de Petri.



**Figura 18. Red de Petri que representa la escritura de una variable compartida. Siendo “a” y “b” la parte de los procesos que acceden a la variable compartida.**

A partir de la gráfica de la red (Figura 18) se obtiene la matriz de incidencia y marcado inicial.

Matriz de Incidencia I =

D1	D2	D3	D4	P1
-1	1	0	0	P1
1	-1	0	0	a

D1	D2	D3	D4	
-1	0	0	1	Sema A
0	1	-1	0	Sema B
0	0	-1	1	P2
0	0	1	-1	b

Vector de Marcado Inicial M0 =

1	0	1	0	1	0
---	---	---	---	---	---

En las tablas siguientes se presentan los valores obtenidos en las mediciones realizadas. Los valores medidos son: cantidad de escrituras, tiempo de ejecución del código sobre el simulador original y sobre el simulador modificado y la relación porcentual entre estas mediciones de tiempo. Además se indica la cantidad de escrituras realizadas en la sección crítica por cada iteración realizada, también se expresa el total de escrituras realizadas por cada hilo sobre el array compartido y la cantidad de instrucciones de procesador ejecutadas por ambos simuladores (original y modificado).

Iteraciones	Escrituras en sección critica (n)	Total de escrituras	Tiempo de ejecución sobre simulador original (ms)	Cantidad de semáforos/disparos ejecutados	Tiempo de ejecución sobre simulador modificado (ms)	Relación %
204800	1	204800	94,823	204800	70,2468	<b>25,92</b>
102400	2	204800	54,888	102400	42,6004	<b>22,39</b>
51200	4	204800	39,476	51200	32,4622	<b>17,77</b>
25600	8	204800	32,821	25600	29,3136	<b>10,69</b>
12800	16	204800	29,493	12800	27,7258	<b>5,99</b>
6400	32	204800	27,592	6400	26,7154	<b>3,18</b>
3200	64	204800	26,641	3200	26,2032	<b>1,64</b>
1600	128	204800	26,225	1600	26,0066	<b>0,83</b>
800	256	204800	26,017	800	25,8008	<b>0,83</b>
400	512	204800	25,913	400	25,8584	<b>0,21</b>
200	1024	204800	25,86	200	25,8322	<b>0,11</b>

Tabla 7. Mediciones realizadas del algoritmo Escritor/Escritor

Nota: La relación porcentual indica la mejora de tiempo obtenida en la ejecución sobre el simulador modificado respecto del simulador original.

En la Tabla 7 se puede apreciar como en la primera fila, la cual ejecuta una primitiva de sincronización (semáforo tomado y liberado) por cada escritura, lo que denominamos un algoritmo fuertemente sincronizado, tiene un porcentaje de mejora de 25,9% en los tiempos de ejecución y la Tabla 8 un 37,32% menos de instrucciones ejecutadas en el simulador modificado. En ambas tablas es evidente como al decrementar el numero de semáforos ejecutados (o sea, incrementar las escrituras realizadas por cada proceso entre

sincronización realizada) disminuye la mejora del procesador modificado hasta niveles inferiores al 1%.

<i>Escrituras en sección crítica (n)</i>	<i>Cantidad de instrucciones ejecutadas por core en simulador original</i>	<i>Cantidad de Instrucciones ejecutadas por core en simulador modificado</i>	<i>Mejora Porcentual</i>
1	28262376	17715276	<b>37,32%</b>
2	15974426	10700916	<b>33,01%</b>
4	11110365	10009605	<b>9,91%</b>
8	8959924	8409504	<b>6,14%</b>
16	7910174	7634964	<b>3,48%</b>
32	7321114	7183494	<b>1,88%</b>
64	7026174	6957364	<b>0,98%</b>
128	6893884	6859474	<b>0,50%</b>
256	6826104	6808894	<b>0,25%</b>
512	6788944	6780334	<b>0,13%</b>
1024	6763227	6758917	<b>0,06%</b>

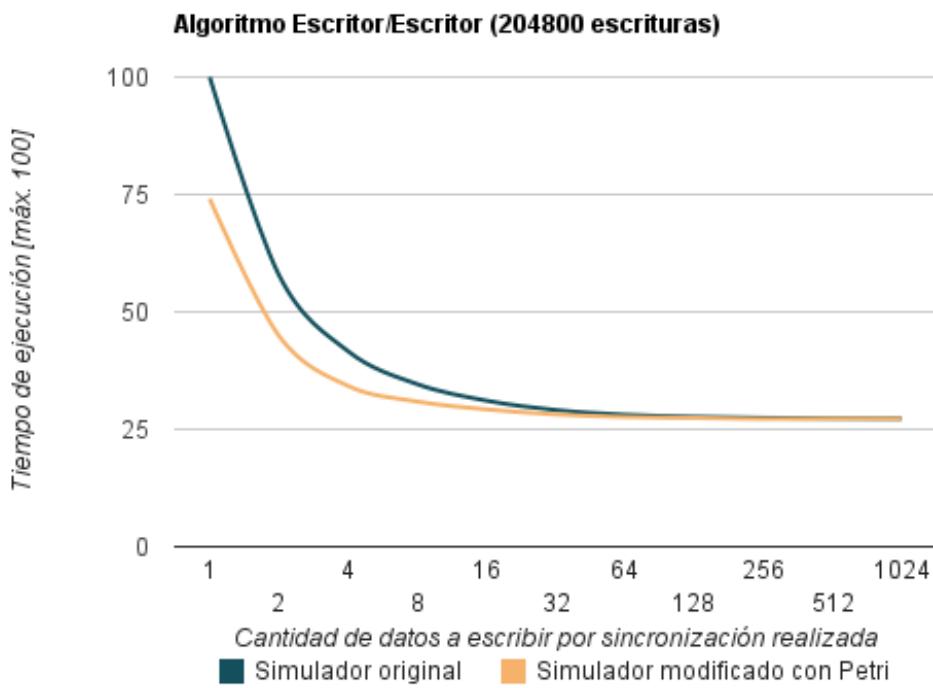
Tabla 8. Mediciones realizadas del algoritmo Escritor/Escritor

### Gráficas comparativas para el simulador Original vs Simulador Modificado, de los resultados obtenidos en la simulación del Algoritmo Escritor / Escritor.

La Figura 19 posee las mediciones de los tiempos empleados para ejecutar el código respecto a la cantidad de escrituras realizadas por sincronización. Claramente puede verse que el tiempo de ejecución disminuye notablemente a medida que disminuye la cantidad de sincronizaciones realizadas, como es lógico de esperar por los estudios realizados anteriormente. **Destacándose el hecho de que existe una disminución de hasta un 25% de los tiempos en la ejecución del código sobre el simulador modificado respecto del original (ver Tabla 7).**

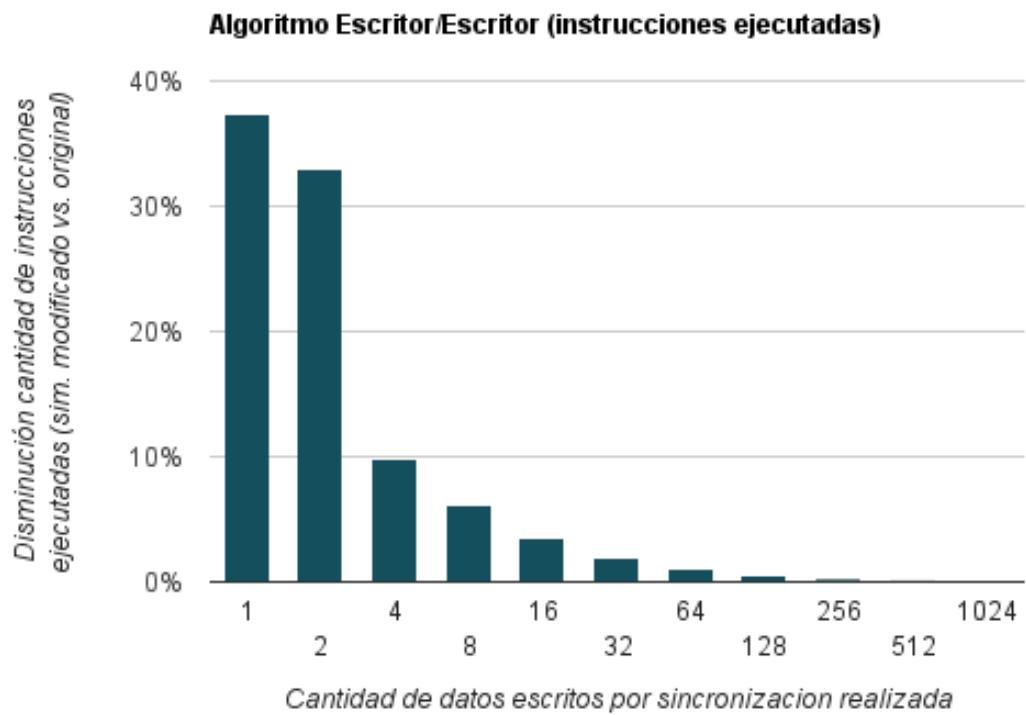
La Figura 20 muestra una relación porcentual de la cantidad de instrucciones ejecutadas comparando el simulador modificado respecto del original. **Se aprecia con claridad la disminución en las instrucciones ejecutadas que posee el simulador modificado respecto al original**, dicha mejora se encuentra más pronunciada para una cantidad de escrituras por sincronización igual a 1y 2, esto es debido al alto impacto producido por la ejecución de una primitiva en cada ciclo o par de ciclos de escritura.

En la Figura 19 es claro como tienden a igualarse las curvas para cantidades de escrituras por sincronización superiores a 64, o tomar valores muy próximos a cero las barras de la Figura 20. Este fenómeno se debe a que los tiempos o instrucciones empleados para ejecutar los semáforos no representa mas de un 1% del tiempo o instrucciones totales, como se había mencionado anteriormente.



**Figura 19. Comparativa entre simulador original y modificado respecto del tiempo de ejecución**

*Nota: el tiempo de ejecución se toma relativamente al máximo tiempo consumido que se le asigna un valor de 100.*

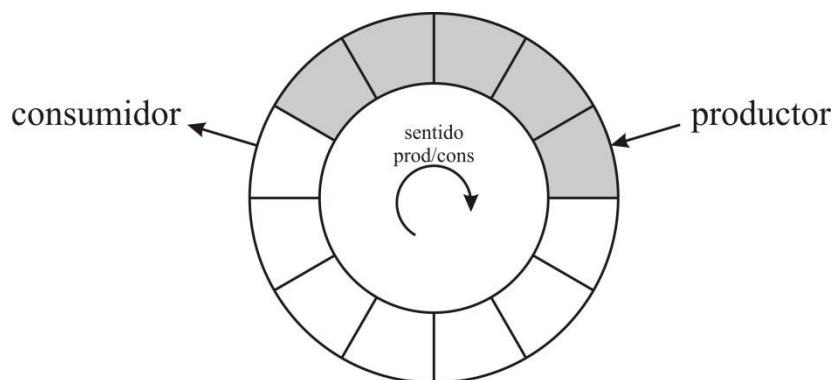


**Figura 20. Diferencia porcentual entre simuladores respecto de las instrucciones ejecutadas**

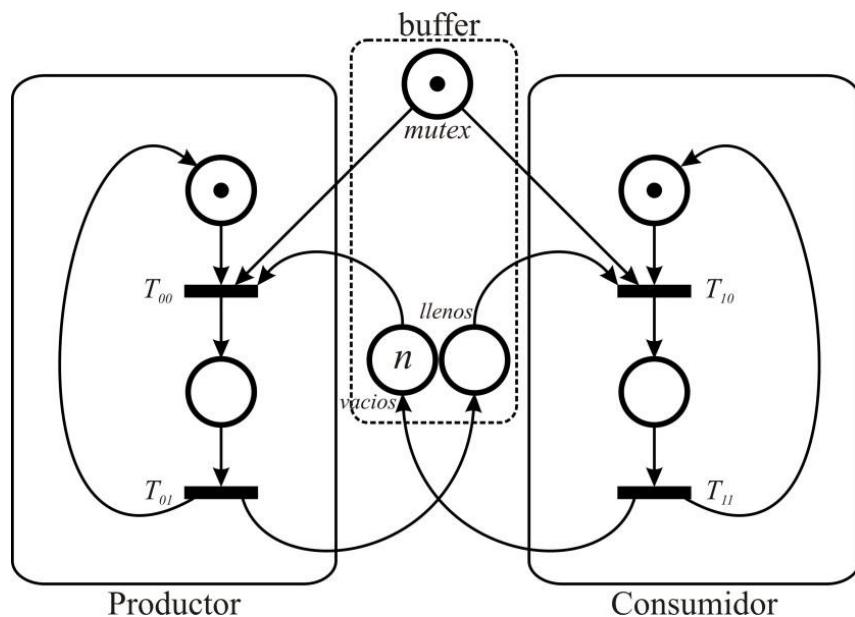
### 6.4.2 Productor / Consumidor

El problema Productor/Consumidor es uno de los ejemplos clásicos de acceso a recursos compartidos que debe arbitrarse mediante algún mecanismo de concurrencia que implemente la exclusión mutua. En este caso el proceso productor genera información que es colocada en un buffer de tamaño limitado y es extraída de este por un proceso consumidor. Si el buffer se encuentra lleno, el productor no puede colocar el dato, hasta que el consumidor retire elementos. Si el buffer se encuentra vacío, el consumidor no podrá retirar elementos hasta que el productor deposite en él.

El hilo principal (el que corre sobre el procesador 0) instancia seis hilos (los cuales se ejecutaban sobre los procesadores 1, 2, 3, 4, 5, 6). Tres de estos hilos pone un dato y los otros tres hilos sacan el dato en un buffer compartido, cuyo tamaño se va a ir modificando (cantidad que se produce/consume) y haciendo distintas simulaciones para obtener así resultados coherentes. A continuación se encuentra la Figura 21 que representa el esquema del *Productor / Consumidor*.



**Figura 21.** Buffer limitado para problema Productor/Consumidor.



**Figura 22.** Red de Petri del modelo Productor/Consumidor

A modo de aclaración de la PN expuesta en la Figura 22, definimos a  $T_{00}$ ,  $T_{01}$ ,  $T_{10}$ ,  $T_{11}$  como las transiciones que forman el modelo del productor/consumidor. El buffer esta

inicializado con n elementos, la plaza denominada *vacíos* indica los lugares vacíos disponibles al proceso productor y la plaza *llenos* le indica al consumidor la cantidad de elementos que le restan por consumir.

A partir de la gráfica de la red se obtiene la matriz de incidencia y marcado Inicial.

Matriz de Incidencia I =

T00	T01	T10	T11	
-1	1	0	0	I0
1	-1	0	0	I1
-1	1	-1	1	I2 (mutex)
-1	0	0	1	I3 (vacíos)
0	1	-1	0	I4 (llenos)
0	0	-1	1	I5
0	0	1	-1	I6

Vector de Marcado Inicial M0 =

1	0	1	10	0	1	0
---	---	---	----	---	---	---

A posterior se encuentran las tablas que reúnen los valores obtenidos en las mediciones, estas contienen: tiempo de ejecución del código sobre el simulador original y sobre el modificado, la relación de mejora porcentual entre estas mediciones de tiempo, también se indica la cantidad de producciones/consumiciones realizadas, la cantidad de semáforos ejecutados y la cantidad de instrucciones de procesador ejecutadas por ambos simuladores (original y modificado).

Cantidad de Prod/Cons	Tiempo de ejecución sobre simulador original(ms)	Cantidad de semáforos/disparos ejecutados	Tiempo de ejecución sobre simulador modificado(ms)	Relación %
500000	198,003	1000000	95,946	<b>51,54</b>
200000	79,203	400000	38,379	<b>51,54</b>
100000	39,603	200000	19,191	<b>51,54</b>
50000	19,702	100000	9,644	<b>51,05</b>
40000	15,762	80000	7,715	<b>51,05</b>
30000	11,822	60000	5,758	<b>51,29</b>
20000	7,882	40000	3,839	<b>51,29</b>
15000	5,912	30000	2,88	<b>51,29</b>
10000	3,942	20000	1,921	<b>51,27</b>

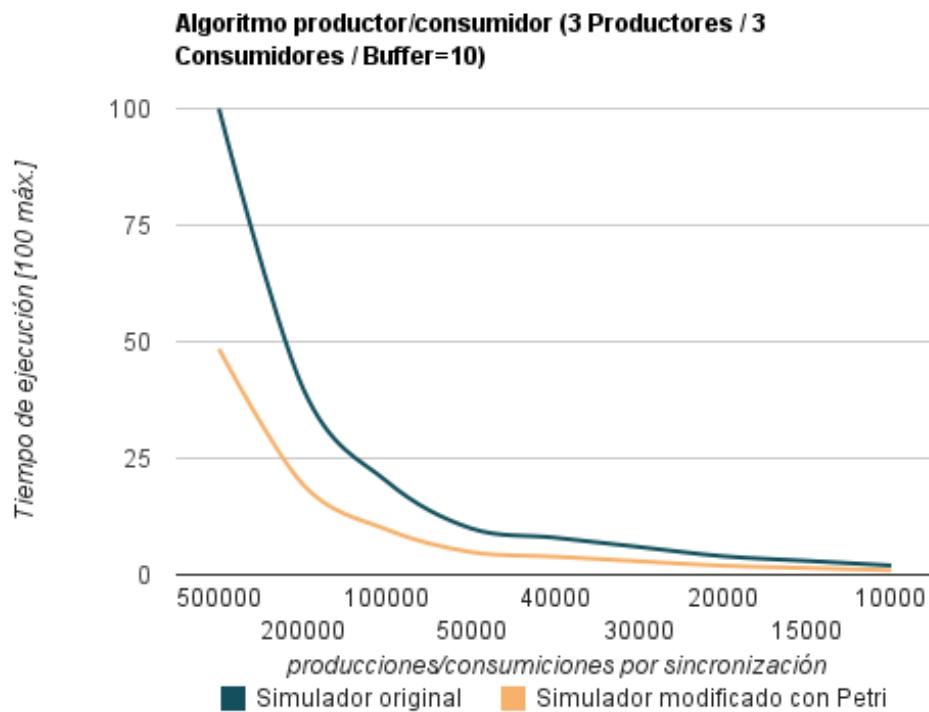
Tabla 9.Mediciones del algoritmo Productor/Consumidor

Nota: La relación porcentual indica la mejora de tiempo obtenida en la ejecución sobre el simulador modificado respecto del simulador original.

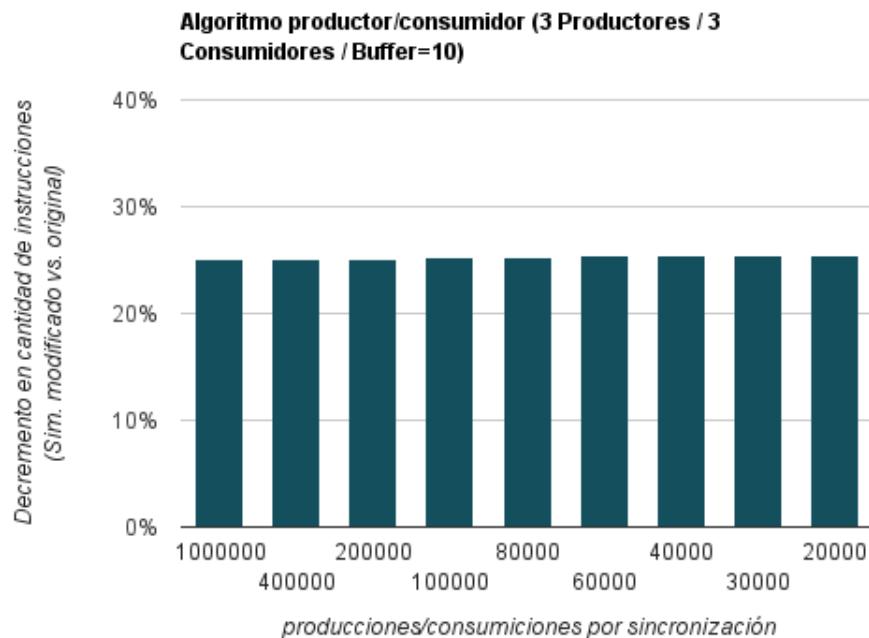
<i>Cantidad de Instrucciones ejecutadas por Core en Simulador Original</i>	<i>Cantidad de Instrucciones ejecutadas por Core en Simulador Modificado</i>	<i>Mejora Porcentual</i>
69689258	52208343	<b>25,08%</b>
27875750	20883343	<b>25,08%</b>
13937914	10441682	<b>25,08%</b>
6918982	5171120	<b>25,26%</b>
5535212	4136942	<b>25,26%</b>
4121373	3073172	<b>25,43%</b>
2747603	2049011	<b>25,43%</b>
2060701	1536922	<b>25,42%</b>
1373816	1024850	<b>25,40%</b>

**Tabla 10. Mediciones del algoritmo Productor/Consumidor**

**Graficas comparativas para el simulador Original vs Simulador Modificado, de los resultados arrojados en la simulación del Algoritmo Productor/Consumidor.**

**Figura 23. Comparativa entre simulador original y modificado respecto del tiempo de ejecución**

*Nota: el tiempo de ejecución se toma relativamente al máximo tiempo consumido que se le asigna un valor de 100.*



**Figura 24. Diferencia porcentual entre simuladores respecto de las instrucciones ejecutadas**

En la Figura 23 se grafican las mediciones de los tiempos empleados para llevar a cabo la ejecución del código, con respecto a la cantidad de producciones y consumiciones realizadas por sincronización, claramente puede verse que los tiempos de sincronización disminuyen a medida que disminuyen la cantidad de sincronizaciones realizadas, en la Tabla 9 se aprecia esta disminución de 50 veces entre la máxima (1000000 de semáforos) y mínima sincronización (20000 semáforos), como era lógico de esperar gracias al estudio realizado con anterioridad. Es importante mencionar que **existe una disminución de mas del 50% de los tiempos empleados en ejecutar el código sobre el simulador modificado (ver Tabla 9) y un 25% menos de instrucciones ejecutadas (ver Tabla 10).**

En ambas curvas de la Figura 23 se tienen valores similares para una cantidad producciones/consumiciones por sincronización igual a 15000, 10000. Debiéndose ese fenómeno a que los tiempos empleados para esas cantidades tienden a asemejarse a los valores de tiempo obtenidos sin emplear sincronización, debido a que el tiempo empleado en sincronizar pasa a ser mínimo por que reducen en 50 veces la cantidad de sincronizaciones realizadas respecto a la máxima sincronización (ver Tabla 9).

En la Figura 24 se muestra una relación porcentual de la cantidad de instrucciones ejecutadas comparando el simulador modificado respecto del original. En esta figura **se aprecia la concreta disminución en la cantidad de instrucciones empleadas en el simulador modificado, siendo este decremento de un 25%.**

#### 6.4.3 Algoritmo de Simulación de una Planta de embalaje (15).

El sistema es una fábrica que consiste en cadenas de montaje que ensamblan piezas a través de cintas transportadoras. Cada cadena de montaje está formada por distintas máquinas para procesar las piezas, almacenes y un robot que recoge y deja las piezas de/en las cintas transportadoras y que carga y descarga máquinas y almacenes. Dependiendo del producto que la fábrica este en ese momento fabricando, cada cadena de montaje sigue un modo distinto de operación.

La Figura 25 muestra una cadena de montaje, el robot recibe una pieza de la cinta A, la carga en la maquina M1 que la procesa de algún modo, y la pieza se almacena temporalmente hasta que pueda cargarse en la maquina M2 para otra operación y entonces se envía a otra cadena de montaje a través de la cinta B.

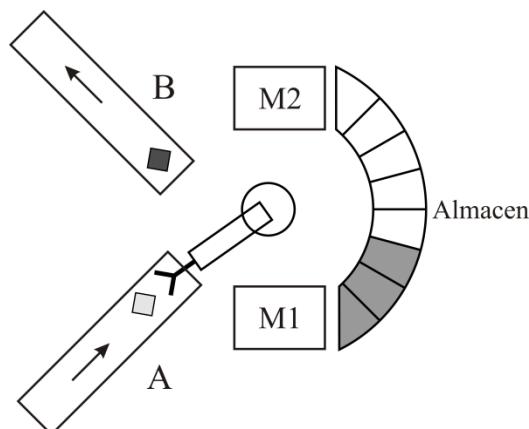
Los lugares de la red de Petri asociada, en la Figura 26, se corresponden con las siguientes operaciones:

- p1: La máquina M1 espera una pieza
- p2: El robot carga la máquina M1 con una pieza recibida desde la cinta A
- p3: La máquina M1 trabajando
- p4: La máquina M1 preparada para descarga
- p5: El robot descarga la máquina M1 y almacena la pieza en el Almacen1
- p6: La máquina M2 trabajando
- p7: La máquina M2 preparada para descarga
- p8: El robot descarga la máquina M2 y envía la pieza a la cinta B
- p9: La máquina M2 espera una pieza
- p10: El robot carga la máquina M2 con una pieza del Almacen1
- A: Piezas preparadas para cogerse desde la cinta A
- B: Piezas en la cinta B
- D: Número de espacios libres en el Almacen1
- E: Número de piezas en el Almacen1

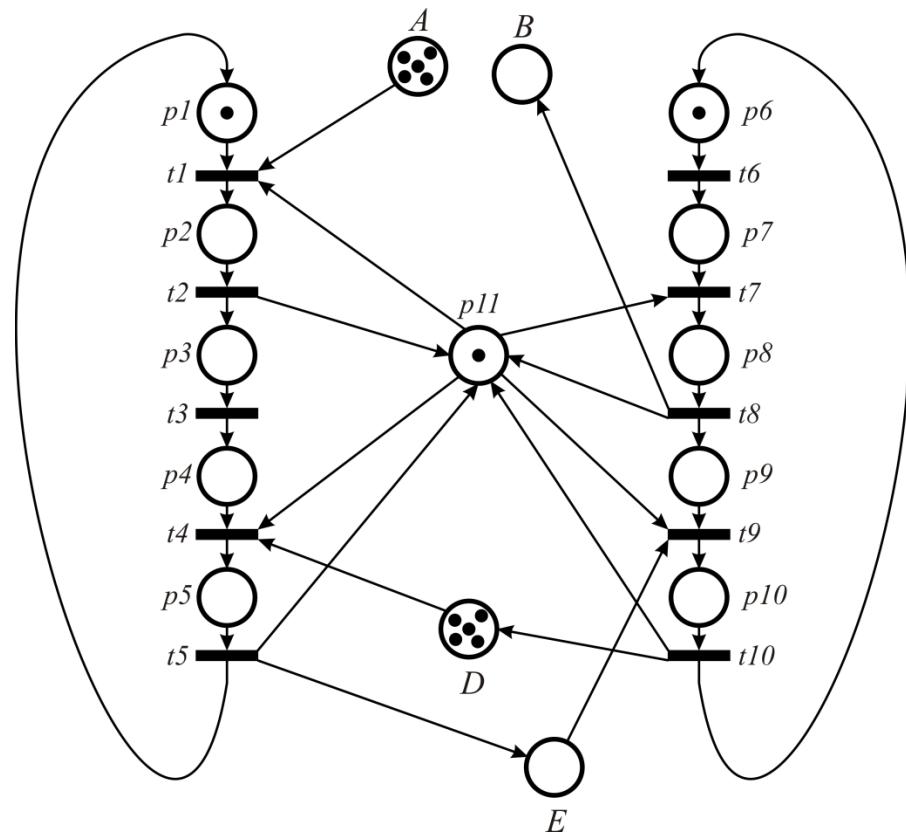
En esta red partimos de la condición de que las máquinas M1 y M2 están esperando a que se les cargue una pieza, y el robot está disponible.

La implementación de este algoritmo se llevó a cabo mediante el empleo de 4 hilos, de los cuales el hilo 0 se encargó de la inicialización de los hilos 1, 2, 3, y los demás hilos ejecutaran distintos disparos para llevar a cabo la ejecución de la red de Petri.

En la Figura 25 se presenta el modelo de la planta de embalaje, mientras que la Figura 26, es la red de Petri que representa a la planta de embalaje.



**Figura 25. Representación del Mecanismo de embalaje**

**Figura 26. Red de Petri de la planta de embalaje**

Es importante destacar que en este algoritmo solo se realizó la sincronización de los hilos.

A partir de la gráfica de la red se obtiene la matriz de incidencia y marcado Inicial.

Matriz de Incidencia I =

t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	
-1	0	0	0	0	0	0	0	0	0	<b>A</b>
0	0	0	0	0	0	0	1	0	0	<b>B</b>
0	0	0	-1	0	0	0	0	0	1	<b>D</b>
0	0	0	0	1	0	0	0	-1	0	<b>E</b>
-1	0	0	0	1	0	0	0	0	0	<b>p1</b>
1	-1	0	0	0	0	0	0	0	0	<b>p2</b>
0	1	-1	0	0	0	0	0	0	0	<b>p3</b>
0	0	1	-1	0	0	0	0	0	0	<b>p4</b>
0	0	0	1	-1	0	0	0	0	0	<b>p5</b>
0	0	0	0	0	-1	0	0	0	1	<b>p6</b>
0	0	0	0	0	1	-1	0	0	0	<b>p7</b>

t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	
0	0	0	0	0	0	1	-1	0	0	<b>p8</b>
0	0	0	0	0	0	0	1	-1	0	<b>p9</b>
0	0	0	0	0	0	0	0	1	-1	<b>p10</b>
-1	1	0	-1	1	0	-1	1	-1	1	<b>p11</b>

Vector de Marcado Inicial M0 =

LOOP	0	ALMACEN	0	1	0	0	0	0	0	0	0	1	0	1
------	---	---------	---	---	---	---	---	---	---	---	---	---	---	---

Las tablas que se encuentran a continuación, muestran los valores obtenidos en la simulación del programa, tiempo de ejecución del código sobre el simulador Original, sobre el simulador Modificado, la relación porcentual entre estas mediciones de tiempo. También se indica el tamaño de loop, la cantidad de semáforos ejecutados y el número de instrucciones de procesador ejecutadas por el Simulador Original y Modificado.

Cantidad de Embalajes	Tiempo de ejecución sobre simulador original(ms)	Cantidad de semáforos/disparos ejecutados	Tiempo de ejecución sobre simulador modificado(ms)	Relación %
1000000	432,002	3000000	104,002	<b>75,93</b>
100000	43,202	300000	10,402	<b>75,92</b>
10000	4,302	30000	1,822	<b>57,65</b>
1000	0,467	3000	0,208	<b>55,46</b>

Tabla 11. Mediciones en algoritmo de planta de embalaje

Nota: La relación porcentual indica la mejora de tiempo obtenida en la ejecución sobre el simulador modificado respecto del simulador original.

Cantidad de Instrucciones ejecutadas por Core en Simulador Original	Cantidad de Instrucciones ejecutadas por Core en Simulador Modificado	Mejora Porcentual
164000251	53000247	<b>67,68%</b>
16400251	5300247	<b>67,68%</b>
1620198	680177	<b>58,02%</b>
171448	75659	<b>55,87%</b>

Tabla 12. Mediciones en algoritmo de planta de embalaje

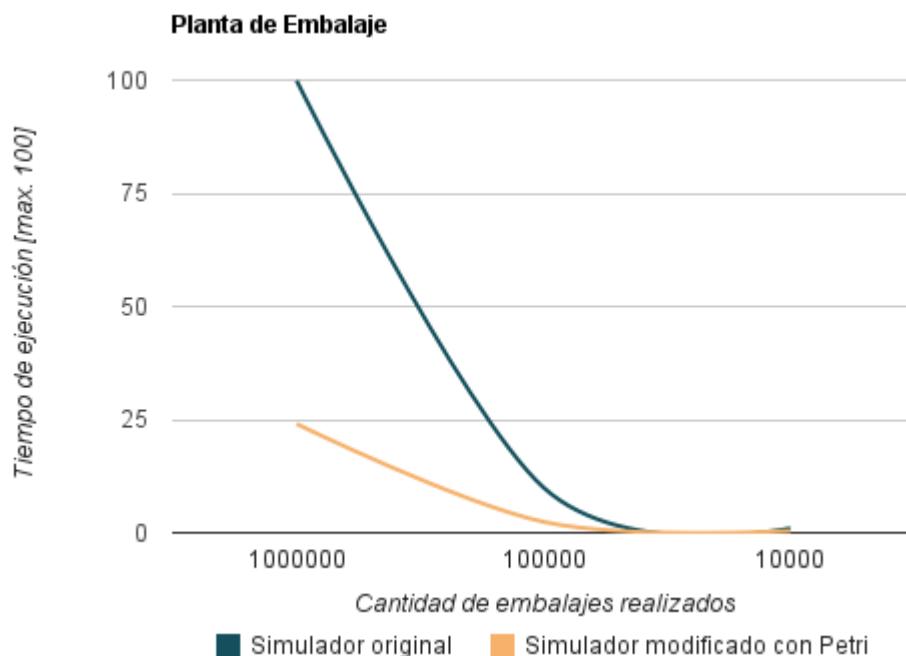
La Figura 27 posee las mediciones de los tiempos empleados para llevar a cabo la ejecución del código, con respecto a la cantidad de embalajes realizados por sincronización. Los tiempos de ejecución disminuyen notablemente a medida que disminuyen la cantidad de sincronizaciones realizadas, debido a una disminución en la cantidad de embalajes. Resultado lógico de esperar debido al estudio realizado con anterioridad. Cabe destacar que **existe una gran disminución en los tiempos de ejecución del código sobre el simulador Modificado (con Petri), siendo este tiempo superior al 55% (ver Tabla**

**11) y la cantidad de instrucciones también disminuye en igual medida (ver Tabla 12).**

En ambas curvas de la Figura 27 obtenemos valores prácticamente iguales para una cantidad embalajes menores a 10000. Debiéndose ese fenómeno a que los tiempos empleados para esa cantidad tienden a asemejarse a los valores de tiempo obtenidos sin emplear sincronización, debido a que el tiempo empleado en sincronizar pasa a ser mínimo comparado con el tiempo del cálculo realizado, debido a que se reducen en gran número la cantidad de sincronizaciones realizadas, esto se aprecia claramente en la columna de los semáforos ejecutados de la Tabla 11, donde se tiene una reducción de hasta mil veces de la primitiva de sincronización ejecutada.

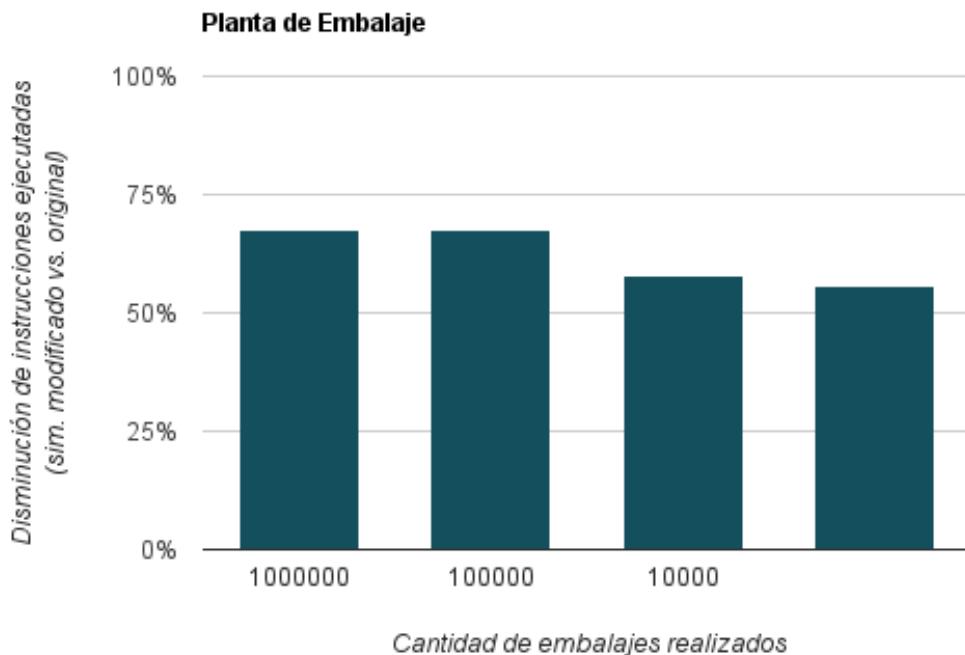
La Figura 28 se muestra una relación porcentual de la cantidad de instrucciones ejecutadas comparando el simulador modificado respecto del original. **Se destaca la marcada diferencia entre la cantidad de instrucciones empleadas para llevar a cabo la ejecución del código que posee el simulador modificado (que emplea Petri), respecto al original,** dicha mejora se encuentra más pronunciada para una cantidad de embalajes iguales a 1000000 y 100000, en donde la disminución de instrucciones ejecutadas alcanza el 67% (ver Tabla 12).

**Gráficas comparativas para el simulador Original vs Simulador Modificado, de los resultados obtenidos en la simulación del Algoritmo de simulación de una planta de embalaje.**



**Figura 27. Comparativa entre simulador original y modificado respecto del tiempo de ejecución**

*Nota: el tiempo de ejecución se toma relativamente al máximo tiempo consumido que se le asigna un valor de 100.*



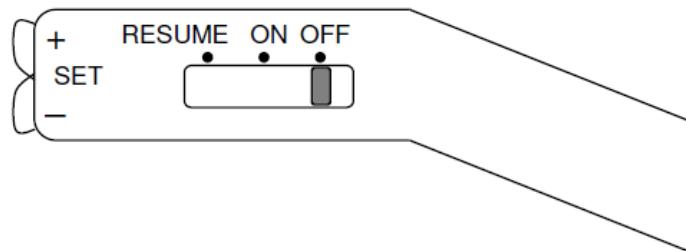
**Figura 28. Diferencia porcentual entre simuladores respecto de las instrucciones ejecutadas**

#### 6.4.4 Algoritmo de control de un mecanismo de velocidad crucero con detector de distancia a objetos para automóviles.

En esta medición se utilizó el ejemplo expuesto en el paper Petri Nets in Software Engineering (16) para poder, mediante simulación, medir la eficiencia del módulo sobre un problema real.

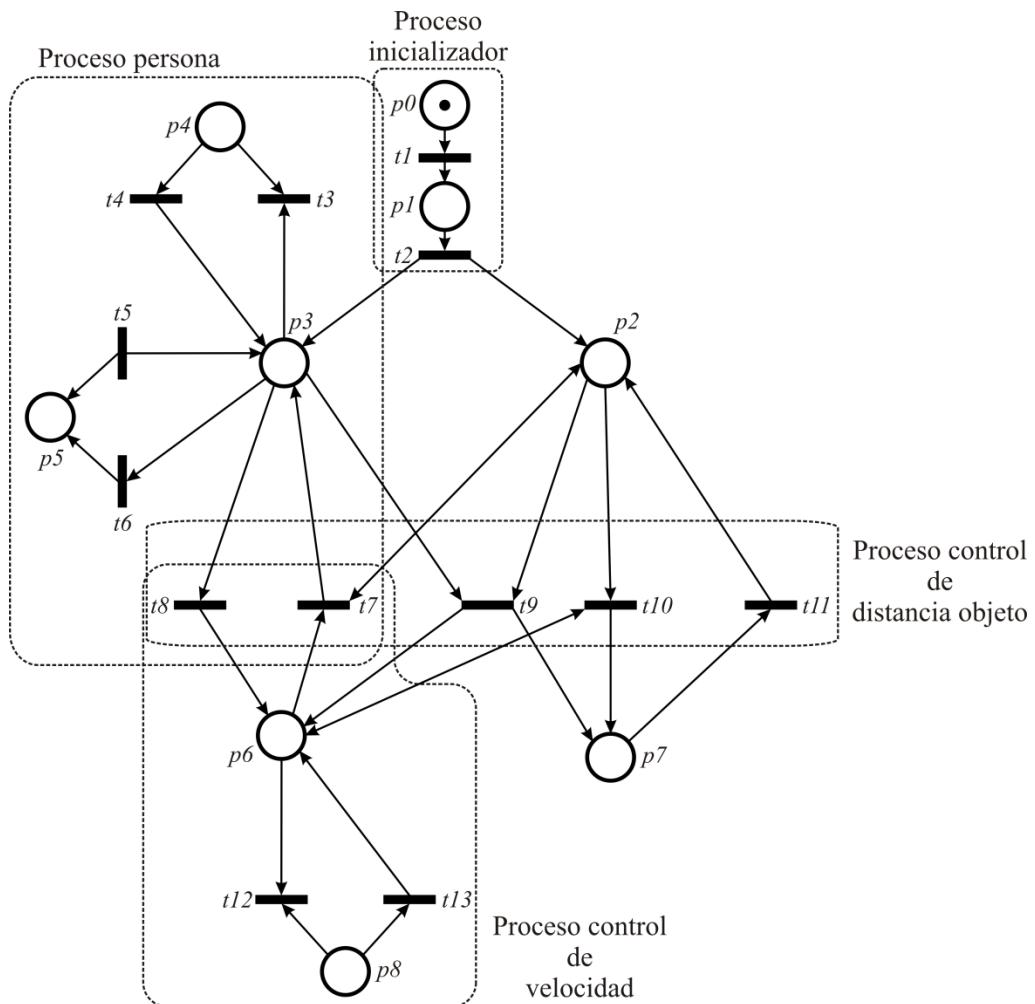
En este algoritmo al igual que en el anterior solo se lleva a cabo la sincronización de los hilos, y no la implementación completa de todo el mecanismo de control, siendo estos dos últimos ejecutados en tiempo real.

Vamos a considerar un sistema de velocidad crucero, con detector de objetos a distancia como se muestra en la Figura 29. En el cual, luego de que el control de velocidad es encendido (ON), la velocidad a la que se desplaza el vehículo es almacenada. Empleando el botón SET (+ o -) el valor de la velocidad crucero, puede ser incrementado o decrementado por 2 km/h. Si el conductor presiona el freno (Brake) el control de velocidad es suspendido, pudiendo este pasar a estado activo mediante RESUME y de nuevo a ON. En el estado de suspendido la velocidad actual es comparada con la velocidad almacenada y un zumbador es activado por un segundo si se supera el valor del control de velocidad. El mecanismo de velocidad crucero, puede ser desactivado moviendo el botón a OFF.



**Figura 29. Interfaz de usuario del mecanismo de Control de Vel. Crucero**  
Fuente: Petri Nets in Software Engineering (16)

En conjunto con el control de crucero se encuentra el mecanismo de advertencia de objeto cercano. Al mismo tiempo que se almacena la velocidad, se activa el mecanismo que mide la distancia al vehículo que se encuentra delante de nosotros. Dicha distancia es comparada con la mínima distancia permitida. Si la distancia medida es menor que la permitida y el sistema de velocidad crucero se encuentra activo, este es suspendido, el vehículo desacelera y el conductor es informado de esto mediante una advertencia (por medio de un LED).



**Figura 30. Red de Petri del Sistema**

Las transiciones con las que cuenta nuestra Red de Petri de la Figura 30 son:

*t1: Encendido*

*t2: Almacenar velocidad*

*t3 - t4: Incrementar velocidad*

*t5 - t6: Decrementar Velocidad*

*t7: Reactivar el mecanismo de Crucero*

*t8: Frenar*

*t9: Distancia Medida menor a la Mínima permitida / Desacelerar*

*t10: Distancia Por debajo de la Mínima*

*t11: Distancia por encima de la Mínima*

*t12 - t13: Control de velocidad Excedido*

Los estados con los que cuenta nuestra red son:

*p0: Apagado (Off)*

*p1: Listo (Ready)*

*p2: Distancia Peligrosa Activa*

*p3: Control Crucero Activo*

*p4: Aumento de velocidad*

*p5: Disminución de velocidad*

*p6: Suspendido*

*p7: Distancia medida por debajo de la mínima*

*p8: Control de velocidad*

La Implementación de este algoritmo en el simulador, se logró, empleando 5 hilos. El hilo 0 hizo la inicialización y la creación de los hilos 1, 2, 3, 4 los cuales se encargaron de ejecutar de forma concurrente todos los disparos de la red. Las mediciones se llevaron a cabo para distinta cantidad de iteraciones de una secuencia de trabajo del control de velocidad.

A partir de la gráfica de la red se obtiene la matriz de incidencia y marcado Inicial.

Matriz de Incidencia I =

<b>t1</b>	<b>t2</b>	<b>t3</b>	<b>t4</b>	<b>t5</b>	<b>t6</b>	<b>t7</b>	<b>t8</b>	<b>t9</b>	<b>t10</b>	<b>t11</b>	<b>t12</b>	<b>t13</b>	
-1	0	0	0	0	0	0	0	0	0	0	0	0	<b>p0</b>
1	-1	0	0	0	0	0	0	0	0	0	0	0	<b>p1</b>
0	1	0	0	0	0	0	0	-1	-1	1	0	0	<b>p2</b>
0	1	-1	1	-1	1	1	-1	-1	0	0	0	0	<b>p3</b>
0	0	1	-1	0	0	0	0	0	0	0	0	0	<b>p4</b>
0	0	0	0	1	-1	0	0	0	0	0	0	0	<b>p5</b>
0	0	0	0	0	0	-1	1	1	0	0	.1	1	<b>p6</b>
0	0	0	0	0	0	0	0	1	1	-1	0	0	<b>p7</b>
0	0	0	0	0	0	0	0	0	0	0	1	-1	<b>p8</b>

Vector de Marcado Inicial M0 =

1	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---

Las tablas que se muestran a continuación, contienen los valores obtenidos en la simulación del programa, tiempo de ejecución del código sobre el simulador Original, so-

bre el simulador Modificado, la relación porcentual entre estas mediciones de tiempo. También se indica el número de iteraciones realizadas, la cantidad de semáforos ejecutados y el número de instrucciones de procesador ejecutadas por ambos simuladores (Original y Modificado).

<i>Iteraciones</i>	<i>Tiempo de ejecución sobre simulador original(ms)</i>	<i>Cantidad de semáforos ejecutados</i>	<i>Tiempo de ejecución sobre simulador modificado(ms)</i>	<i>Relación %</i>
100000	145,472	700000	60,354	<b>58,51</b>
10000	14,592	70000	6,275	<b>57</b>
1000	1,458	7000	0,629	<b>56,86</b>
100	0,147	700	0,0642	<b>56,32</b>
10	0,016	70	0,00792	<b>50,5</b>

Tabla 13. Mediciones del control de crucero

Nota: La relación porcentual indica la mejora de tiempo obtenida en la ejecución sobre el simulador modificado respecto del simulador original.

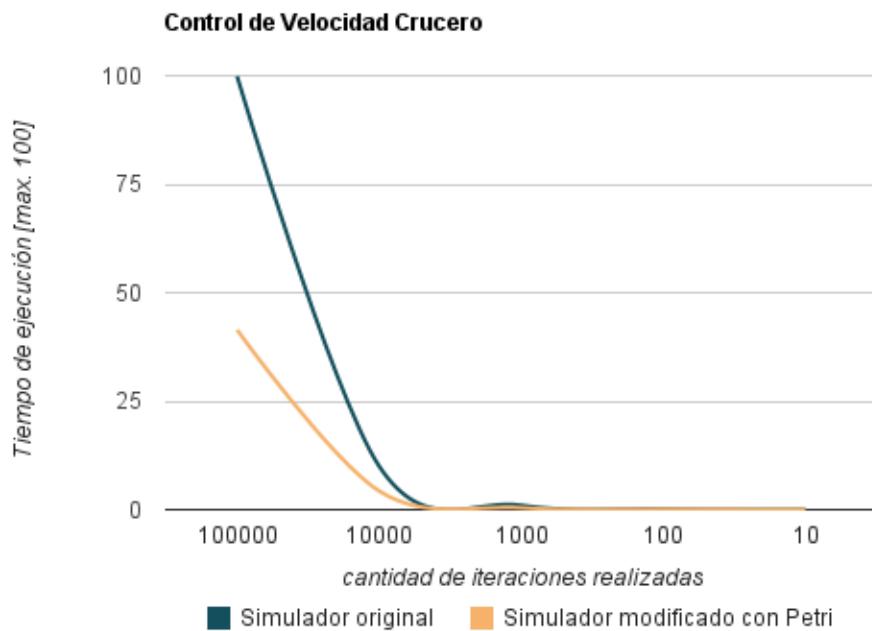
<i>Cantidad de Instrucciones ejecutadas por core en simulador original</i>	<i>Cantidad de instrucciones ejecutadas por core en simulador modificado</i>	<i>Mejora porcentual</i>
46913745	18217625	<b>61,17%</b>
4688523	1867513	<b>60,17%</b>
468213	186813	<b>60,10%</b>
46953	18833	<b>59,89%</b>
4783	2013	<b>57,91%</b>

Tabla 14. Mediciones del control de crucero

**Gráficas comparativas para el simulador Original vs Simulador Modificado, de los valores obtenidos en la simulación del Algoritmo de control del Mecanismo de control de velocidad crucero con detección de vehículos a distancia.**

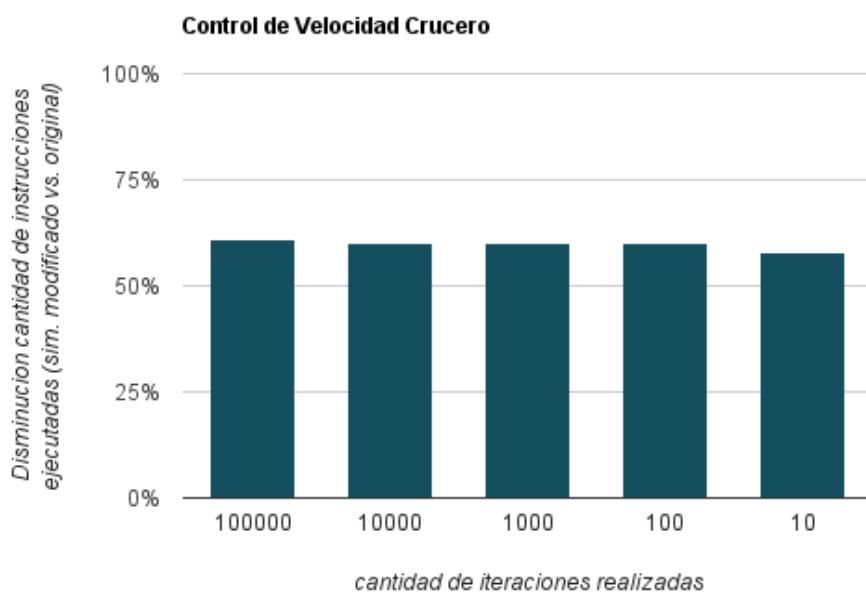
En la Figura 31 se encuentran las mediciones de los tiempos empleados para llevar a cabo la ejecución del código, claramente se observa que estos tiempos disminuyen a medida que es menor la cantidad de ejecuciones (iteraciones), debido a que es menor la cantidad de sincronizaciones realizadas, **obteniéndose con el simulador modificado (empleando Petri) una disminución superior al 50% en los tiempos de ejecución (ver Tabla 13).**

En la Figura 32 se muestra una relación porcentual de la cantidad de instrucciones ejecutadas comparando el simulador modificado respecto del original. **Puede apreciarse una importante disminución en la cantidad de instrucciones para llevar a cabo la ejecución del código que posee el simulador modificado (que emplea Petri), respecto al original, siendo esta disminución de entre 57% y 61% en la cantidad de instrucciones ejecutadas (ver Tabla 14).**



**Figura 31. Comparativa entre simulador original y modificado respecto del tiempo de ejecución**

*Nota: el tiempo de ejecución se toma relativamente al máximo tiempo consumido que se le asigna un valor de 100.*



**Figura 32. Diferencia porcentual entre simuladores respecto de las instrucciones ejecutadas**



## 7 CONCLUSIÓN

---

En este proyecto, en primera instancia se realizó un estudio sobre los costos (tiempos) a nivel de procesamiento de la sincronización de procesos. A posteriori nos dedicamos al análisis y comprensión del simulador SESC. En dicho estudio realizado sobre los datos arrojados por el simulador (Tiempo de ejecución, Ciclos de reloj empleados, Cantidad de instrucciones ejecutadas por core, Tipo de instrucción ejecutadas, Unidades funcionales utilizadas, Cantidad de Hits y Miss por cache) obtuvimos la información suficiente que **nos permitió comprender el problema de eficiencia de la sincronización entre procesos y determinar una solución a este**. Dicha solución es instanciar un procesador de Petri dentro de una nueva cache compartida por todos los procesadores, para la sincronización de procesos en un sistema multicore, **desarrollando así un nuevo concepto de Procesador y una nueva Arquitectura**.

Esta solución implementada en el simulador **permitió obtener una disminución por encima del 25% (alcanzando valores del 75%) en los tiempos de ejecución de procesos fuertemente sincronizados, reduciendo también el número de instrucciones ejecutadas en más de un 20% respecto de los procesadores estándares simulados**. Estas mejoras han sido demostradas en las Simulaciones y Mediciones realizadas, siendo importante destacar que se evidencian especialmente en los programas que emplean fuerte sincronización entre procesos. Además remarcando el hecho de que el nuevo procesador es más rápido ya que tiene mejor respuesta en tiempo real a la hora de sincronizar.

Por otro lado, es importante mencionar que la realización de los programas para que puedan ser ejecutados en el nuevo procesador son de fácil programación, ya que una vez realizado el grafo de la red de Petri se pueden obtener las matrices mediante la utilización de cualquier simulador de redes de Petri.

Hay que destacar que si en un programa, por ejemplo se tratara de múltiples escritores o múltiples productores/consumidores la sobrecarga impacta en los recursos de memoria y procesador pero no sobre el algoritmo de Petri puesto que este solo atiende a los requerimientos de sincronización y exclusión mutua, es decir que la sobrecarga sobre el modulo está dada por la cantidad de disparos por unidad de tiempo.

### 7.1 Comentarios Adicionales

En la realización de este proyecto se cumplieron las condiciones y objetivos planteados, superándose ampliamente las expectativas sobre el funcionamiento de la implementación y obteniéndose excelentes resultados (destacados anteriormente).

A principios del 2011 se comenzó con el desarrollo de este proyecto finalizándose a fines del mismo año. La realización de este nos resultó enriquecedora permitiéndonos aplicar conceptos vistos en distintas asignaturas de la carrera, ampliar los conocimientos de C, C++, mejorar las metodologías de trabajo en equipo, profundizar los conocimientos relacionados con la Arq. De Computadoras, y exponer nuestro Proyecto Integrador en un Workshop en el congreso CRUNIC 2011.

## 8 TRABAJOS FUTUROS

---

Desarrollo de un IPCore que implemente el algoritmo de Petri. En la actualidad se está llevando a cabo en el Laboratorio de Arquitectura de Computadoras de la FCEFyN-UNC.

Realizar la extensión del algoritmo a Redes de Petri con brazos inhibidores y de reset.

Realizar la extensión del algoritmo citado a Redes de Petri temporales y coloreadas.

La simulación, implementación y experimentación de estas propuestas para comprobar las mejoras en el desempeño de la ejecución de algoritmos fuertemente sincronizados en sistemas multicore.



## 9 ANEXO I

---

### 9.1.1 ¿Qué es un superescalar fuera de orden (out-of-order) pipeline?

Podemos suponer un procesador como una caja negra donde se ejecuta una instrucción en cada ciclo de reloj, pero los procesadores actuales pueden tener varias instrucciones ejecutándose internamente en las diferentes etapas de un pipeline.

En la clasificación de Flynn, un procesador superescalar es un procesador de tipo MIMD (múltiple instrucción múltiple data).

La arquitectura superescalar utiliza el paralelismo de instrucciones además del paralelismo de flujo, éste último gracias a la estructura en pipeline. La estructura típica de un procesador superescalar consta de un pipeline con las siguientes etapas:

1. Lectura (*fetch*).
2. Decodificación (*decode*).
3. Lanzamiento (*dispatch*).
4. Ejecución (*execute*).
5. Escritura (*writeback*).
6. Finalización (*retirement*).

En un procesador superescalar, el procesador maneja más de una instrucción en cada etapa. El número máximo de instrucciones en una etapa concreta del pipeline se denomina grado, así un procesador superescalar de grado 4 en lectura (*fetch*) es capaz de leer como máximo cuatro instrucciones por ciclo. El grado de la etapa de ejecución depende del número y del tipo de las unidades funcionales.

Un procesador superescalar puede tener unidades funcionales independientes de los siguientes tipos:

1. Unidad aritmético lógica (ALU)
2. Unidad de lectura/escritura en memoria (Load/Store Unit)
3. Unidad de coma flotante (Floating Point Unit)
4. Unidad de salto (Branch unit)

La razón de la utilización de una arquitectura Superescalar, se debe a que esta se encuentra muy estudiada y desarrollada de tal forma que sus ventajas y desventajas son conocidas, permitiendo así la existencia de un gran campo de investigación.

*Limitaciones de la misma:*

- *La memoria.* El acceso a memoria es cada vez más lento en relación a la velocidad del procesador.
- *Las dependencias.* Las dependencias entre instrucciones restringen el grado de paralelismo que puede explotar el procesador.
- *Los retardos en las interconexiones.* Al mejorar el factor de integración de la tecnología, disminuye el retardo de la lógica. Sin embargo, el retardo de las interconexiones entre diversos bloques de un mismo chip apenas se reduce, por lo que estos retardos tienen un peso cada vez mayor.
- *Consumo/dispersión de energía.* Las progresivas reducciones de la tensión de alimentación no son suficientes para paliar este problema. Aportaciones desde otras áreas tales como la micro arquitectura empiezan a ser vitales.

### 9.1.2 ¿Cómo modela esto SESC?

En SESC, las instrucciones son ejecutadas en un módulo del simulador, el cual emula un ISA de un MIPS. El SESC emula las instrucciones binarias de un programa en orden, el módulo que genera las instrucciones es el MINT, un emulador de MIPS. Este retorna objetos instrucción al simulador de tiempos. Los objetos contienen la dirección de la instrucción, la dirección de load o strore en la memoria, los registros fuente y destino y las unidades funcionales utilizadas por la instrucción (no contiene valores de registros o posiciones de memoria utilizados). Toda esta información es la utilizada por el simulador para determinar el tiempo empleado por la instrucción que sale del pipeline.

La ventaja de dividir SESC entre emulación y simulación es que se vuelve más sencillo de programar y depurar debido a que la ejecución y la medición de tiempos están separadas.

### 9.1.3 Análisis detallado del simulador

Describiremos como una instrucción en una aplicación fluye a través del simulador.

#### 9.1.3.1 Tipos de instrucciones

##### 9.1.3.1.1 Instrucciones Estáticas (Static Instructions)

Un ejemplo de este tipo de instrucción es **add r1,r2,r3**. Esto es implementado en la clase Instrucción (esesc/src/Libll/Instruction.cpp). Las instrucciones son creadas durante la inicialización de SESC. Una función decodifica cada una de las instrucciones de la aplicación para que sea utilizada internamente.

La representación interna contiene información para ejecutar rápidamente la instrucción.

El objeto instrucción contiene: registro fuente, tipo de instrucción (salto, load, store, cálculos aritméticos enteros o de punto flotante), un puntero a una función para ejecutar la instrucción específica (como una suma de punto flotante), un puntero a la siguiente instrucción a ejecutar y un puntero al destino de un salto si existiese.

##### 9.1.3.1.2 Instrucciones Dinámicas (Dynamic Instructions)

Son instancias específicas de instrucciones estáticas. Cuando cada instrucción estática es ejecutada una instrucción dinámica es creada. Por ejemplo, la instrucción dinámica que hace referencia a la estática **add r1,r2,r3**, sería r1=10, r2=8, y el procesador se en-

cuenta en el ciclo de reloj número 32921. Las instrucciones dinámicas son implementadas en la clase DInst(esesc/src/libcore/DInst.cpp).

Las instrucciones dinámicas son referenciadas teniendo en cuenta las dependencias entre las mismas. Las instrucciones mantienen referencia a las dependencias de las instrucciones anteriores a través de un puntero. También contienen variables para el seguimiento del estado de ejecución como emitida, ejecutada, o finalizada; a que CPU corresponde la instrucción, el número de secuencia y que recursos necesita la instrucción dinámica (como un multiplicador de punto flotante, etc).

### 9.1.3.2 Emulación

La emulación es controlada por la clase ExecutionFlow (se encuentra en esesc/src/Libll/). La interfaz hacia el nivel superior es a través de la función executePC(). La función executePC() en ejecución llama a exeInst(), la cual realiza algunos controles como por ejemplo si la dirección de la instrucción es válida, y luego la ejecuta. exeInst() retorna cero si ninguna instrucción pudo ser ejecutada, un valor distinto de cero si la ejecución de una instrucción fue exitosa (si se trató de un load/store el valor retornado corresponde a la dirección virtual del dato accedido). La ejecución de executePC() concluye con el retorno de un puntero a la instrucción dinámica ejecutada (DInst).

Dentro de la clase ExecutionFlow hay un método que ejecuta instrucciones en **rabbit mode**, este modo solo ejecuta las instrucciones (emulación) sin simular los tiempos (cada instrucción es solo emulada, siendo 1000 veces más rápida la simulación que en el **modo full**, en el cual se emulan y simulan las instrucciones). Este modo es seleccionado a través de un parámetro al ejecutar el simulador.

### 9.1.3.3 Pipeline

En SESC, el objeto GProcessor, un procesador genérico (se encuentra en esesc/src/libcore/Gprocessor.cpp). Tiene como responsabilidad coordinar la interacción entre las diferentes etapas del pipeline. Esta relación con los demás componentes se puede apreciar en la Figura 33.

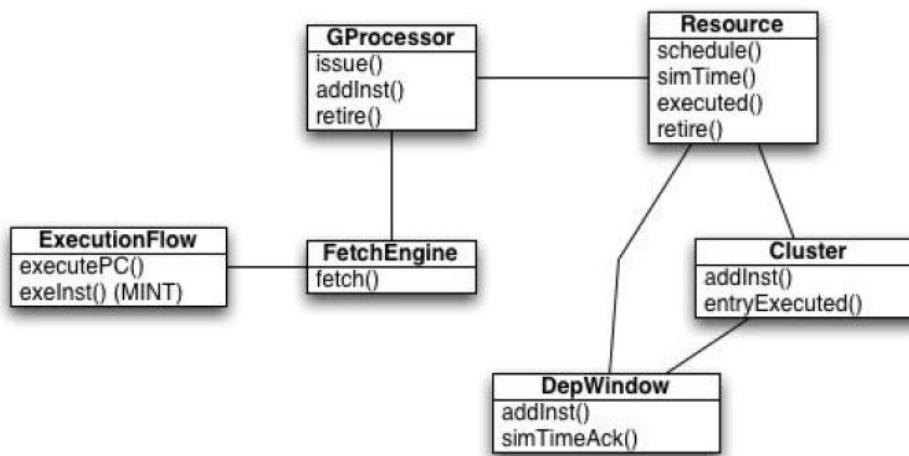


Figura 33. Relación entre las clases del procesador

La interfaz con el nivel superior al objeto GProcessor es la función advanceClock(). La función advanceClock() mueve la instrucción por cada etapa del pipeline en cada pulso de reloj. Esto lo realiza buscando inicialmente una instrucción y la coloca en la cola de instrucciones, luego emite una instrucción desde la cola hacia la ventana de planificación.

Para la planificación y ejecución hay dos módulos, uno para enteros y otro para punto flotante, y cada uno tiene su propia ventana de planificación (Tomasulo (17)). Ver más adelante planificación y ejecución. Finalmente GProcessor ejecuta retire(), la cual retira las instrucciones ejecutadas desde el buffer de reordenamiento (ROB).

#### 9.1.3.3.1 Fetch/Decode

Es la primera etapa del pipeline. En él, las instrucciones son tomadas desde la cache de instrucciones (en la configuración típica se leen 4 instrucciones por ciclo). En esta etapa se predicen hacia donde debe tomarse los saltos condicionales. La clase del simulador que realiza estas tareas es la de FetchEngine ( contenida en esesc/src/libcore/). La interfaz con la clase se realiza a través del método fetch(). Este método trae las instrucciones desde la cache y modela el predictor de saltos. La función fetch() interactúa con ExecutionFlow, a través del método ExecutePC(), para tomar la dirección de la siguiente instrucción a buscar en la cache. Una vez que obtiene la dirección hace el pedido a la cache para obtener la instrucción.

Al finalizar la búsqueda del conjunto de instrucciones, son enviadas a la etapa de decodificación, esta etapa transforma las instrucciones del formato ISA a un formato interno para generar el objeto Instrucción (instrucción estática).

#### 9.1.3.3.2 Predictor de salto

SESC soporta diferentes predictores de salto. La selección y el tamaño se seleccionan en el momento de ejecución. Desde que la unidad de predicción esta lista en la etapa de búsqueda, es manejada por la clase FetchEngine. Si la instrucción es un salto llama al método processBranch(), el cual hace la predicción. Si la predicción realizada es incorrecta, la función processBranch() modela el vaciado del pipeline marcando el error del salto. Llamadas sucesivas de búsqueda de instrucción utilizan fakeFetch(), el cual busca instrucciones en el path incorrecto que causó la predicción de salto errónea. Estas instrucciones serán marcadas como instrucciones falsas.

Una vez ejecutado el salto mal predicho, el predictor actualiza el objeto FetchEngine y restaura el correcto curso de ejecución.

#### 9.1.3.3.3 Emisión y planificación

Durante la etapa de emisión, las instrucciones son tomadas desde la cola de instrucciones donde estaban almacenadas, y enviadas a planificadores de ventanas individuales en un cluster particular (cluster se denomina a un contenedor de recursos e instrucciones instanciados para cada procesador). La planificación (schedule) hace referencia a cuando los operandos de entrada para una instrucción están listos, y la instrucción es planificada para ejecución. Hasta la planificación todas las instrucciones avanzan en orden (in-order). En la planificación y la ejecución, las instrucciones son ejecutadas fuera de orden (out-of-order), o sea, como están listas para ejecutar. Luego, en la etapa de retiro, las instrucciones son retiradas en orden.

La función issue() en el objeto GProcessor toma un número configurable de instrucciones de la cola de instrucción e intenta ponerlas en la cola del planificador de cada cluster. issue() invoca a addInst() para cada instrucción, si esta falla, issue() retorna y tratará nuevamente en el siguiente ciclo de emitir esa instrucción.

El método addInst() corrobora varias cosas antes de confirmar que la instrucción puede ser emitida. Primero chequea que hay disponibilidad en el buffer de reordenamiento. Luego controla que exista un registro de destino libre. En tercer lugar, este verifica el espacio en la ventana de planificación del cluster. Finalmente, basado en los recursos específicos que requiere la instrucción realiza otros controles. Por ejemplo, para loads, este controla que el máximo número de loads no haya sido excedido. Para saltos, controlará

que el máximo número de saltos pendientes no haya sido alcanzado. Este control es realizado llamando a `schedule()` del objeto `Resource` (contenido en `esesc/src/libcore`) específico para este tipo de instrucción.

Si la verificación de la función anterior en la clase `GProcessor` es correcta, invoca al método `addInst()` para el cluster específico en el cual se ejecutará posteriormente. Finalmente una entrada es agregada al final del buffer de reordenamiento.

Para manejar la dependencia entre instrucciones cuando el registro destino de una instrucción es el fuente de otra, cada `Cluster` contiene un objeto ventana de dependencia (`DepWindow` en `esesc/src/libcore/`). La función `addInst()` en `Cluster` (que se encuentra en `esesc/src/libcore/`) luego llama a la función `addInst()` en `DepWindow` asociada con el Cluster. La clase `Cluster` tiene la responsabilidad de llevar a cabo el algoritmo de Tomasulo (17).

En `DepWindow`, se mantiene una tabla donde se mapean instrucciones a los registros destinos. La tabla solo contiene la referencia de la última instrucción a escribir a cada registro. Cuando `addInst()` es llamada, el registro fuente de la instrucción que será planificada se busca en la tabla. Esto encuentra las instrucciones, si hay alguna, que producen los operandos de entrada para la instrucción. Cuando una instrucción termina la ejecución, si la entrada en la tabla todavía mapea a esta instrucción, es borrada.

Considere el siguiente fragmento de código:

```
1: Id          R1, 0x1000 ; load memory address 0x1000  
2: add        R1, R1, R2
```

En este ejemplo, la instrucción 2 es dependiente de la instrucción 1. La función `addSrc()` es llamada en la instrucción 1 con el parámetro de la instrucción 2. Luego se explicara como 1 despierta a 2 luego de haber concluido su ejecución. También `DInst` 2 marca que depende de una instrucción.

En el caso de que las instrucciones son independientes, estas se planifican para ejecución. Esto es realizado mediante la generación de un evento de retorno de llamada para ejecutar la función `simTime()` de `Resource` en un cierto número de ciclos. El delay depende solo de la penalización fijada en la planificación y del número de instrucciones que están listas para ejecución en el recurso específico, cada recurso puede solo ejecutar un pequeño número de instrucciones por ciclo.

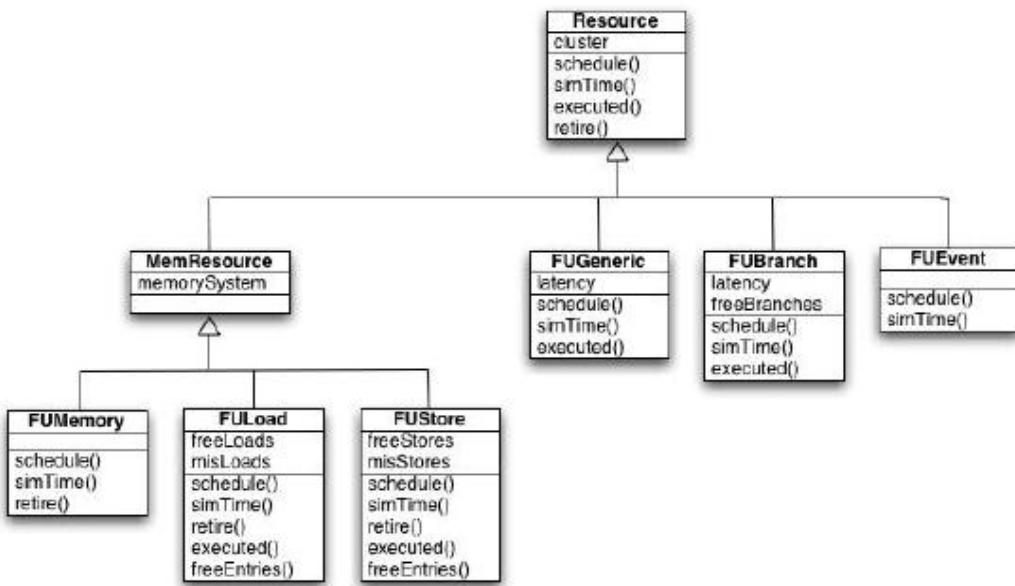
Finalmente, el operando de salida en la tabla es apuntado a la instrucción planificada. Luego, futuras instrucciones que utilizan ese operando de salida se marcarán como dependientes.

#### 9.1.3.3.4 Ejecución

El objeto `DepWindow` planifica una instrucción para ejecución o setea punteros a las instrucciones que debe ejecutar primero.

De cada instrucción planificada debemos tener en cuenta los recursos que va a utilizar para su ejecución, esto está previsto en cada subclase de `Resource`, para cada tipo de instrucción, como muestra en la Figura 34. Cada subclase define su método `simTime()`, el cual simula la ejecución de la instrucción. La ejecución toma un ciclo para los saltos y operaciones enteras, varios ciclos para punto flotante y cientos para las operaciones de memoria.

Una vez concluida `simTime()` se invoca a `executed()` en el objeto `Resource`, un método que retorna si la ejecución de la instrucción fue concluida. Para loads, `FULoad` envía el load a la cache. Para stores, planifica un callback a la función `execute()` (Los stores son enviados a la cache en la etapa de retiro, desde esta etapa pueden ser dados de baja solo por un salto erróneo). Para todas las demás instrucciones el callback es realizado a la función `execute()`.

**Figura 34. Jerarquía de Recursos.**

La función `executed()` llama a `entryExecuted()` para el correspondiente Cluster de ese `Resource()`. `entryExecuted()` llama a `simTimeAck()` en el objeto `DepWindow`, la cual marca a la `DInst` como ejecutada, para así poder retirarla posteriormente. Luego controla si la entrada para el registro destino en la tabla todavía apunta a la `DInst`, de ser así es eliminada. Finalmente la instrucción controla si hay alguna instrucción dependiendo de ella.

#### 9.1.3.3.5 Finalización (Retirement)

En esta etapa de un procesador fuera de orden, las instrucciones que ya fueron ejecutadas son removidas de la cabecera del buffer de reordenamiento en el orden de ejecución del programa original. Típicamente, 3 o 4 instrucciones son retiradas por ciclo. Las instrucciones de store no son enviadas a la cache hasta llegar a esta etapa. Si la cache no acepta el store, la instrucción no puede ser retirada de la ROB.

En SESC, el objeto `GProcessor` llama a `retire()` en cada ciclo. Ese método toma la `DInst` de la cima de la ROB y controla si puede ser retirada, esto se realiza testeando la bandera de ejecución y luego se llama a `retire()` del objeto `Resource`, la cual retorna si la instrucción puede o no ser removida.

#### 9.1.3.4 Cache

Una caché es un componente que almacena datos utilizados por el procesador para que los futuros requerimientos a esos datos puedan ser servidos más rápidamente. Generalmente son datos temporales.

La idea de duplicación de datos se basa en que los datos originales son más costosos de acceder (dato contenido en la memoria RAM) en tiempo con respecto a la copia en memoria caché.

Si se solicitan los datos contenidos en el caché, estos son accedidos rápidamente; de lo contrario, los datos deben ser recomputados o tomados de su ubicación original, lo cual implica un tiempo de acceso mucho mayor al tiempo de acceso de un dato contenido en la cache.

Hay varios niveles de cache:

- L1: Una cache de datos y una de instrucciones.
- L2: Es de mayor tamaño y más lenta que L1, usualmente compartida por todos los procesadores.
- Los niveles inferiores son siempre más lentos y tienen un mayor tamaño.

SESC modela los diferentes tipos de cache según los siguientes parámetros:

- Tamaño.
- Latencia de acierto y errors (Hit & Miss).
- Políticas de reemplazo.
- Cache-line sizes
- Asociatividad.

La implementación es necesariamente compleja. Cada GProcessor tiene un objeto GMemorySystem (esesc/src/libcore). Este objeto crea la jerarquía de cache y sirve como adaptador entre el GProcessor y los niveles más altos de Cache.

Cuando el GProcessor necesita interactuar con el GMemorySystem, este crea un objeto MemoryRequest. Existen objetos DMemoryRequest (para datos) e IMemoryRequest (para instrucciones). Además los DMemoryrequest tienen un atributo el cual indica si el acceso es una lectura o escritura. Los MemoryRequest son creados a través de los métodos singleton DMemRequest::create() o IMemRequest::create(). Estos métodos toman como parámetro los objetos DInt() al cual están asociados los requerimientos, el objeto GMemorySystem por el cual pasa la solicitud y el tipo de operación (lectura o escritura).

La función create() aloca un nuevo objeto MemRequest, lo inicializa y luego llama a access() en la Cache de nivel más alto. Cada una de estas funciones usa un retorno para invocar doRead() o doWrite() en el próximo ciclo disponible de Cache, para modelar los puertos de la memoria cache. Si la lectura o escritura es un acierto en doRead() o doWrite() respectivamente, la función goUp() es llamada en MemRequest. Esta retorna el MemRequest al GProcessor. Por otro lado, la función que maneja los misses es llamada, la cual invoca a goDown() en el MemoryRequest para enviarlo al siguiente nivel de la jerarquía.

El aspecto más importante es que todos los tipos de Caches y Buses heredan de una clase común, MemObj, la cual define una interfaz común que consiste de access(), returnAccess() y otras funciones de menor importancia. returnAccess() es llamada por goUp() cuando un acceso retorna a un nivel superior de cache desde niveles inferiores. Esta interfaz permite herencia de cache totalmente configurable. Cada subtipo de cache tiene diferentes maneras de manejar los requerimientos internamente, siempre y cuando se ajuste a la interfaz para cache de niveles inferiores y superiores.

En un sistema multiprocesador, en el nivel más bajo, cada cache del procesador y la memoria principal están conectados.

#### 9.1.3.5 Llamadas a sistema

SESC no provee un sistema operativo, es por esto que el simulador debe capturar las llamadas a sistema y llevarlas a cabo en nombre de la aplicación. Para cada llamada estándar al sistema, SESC la transforma a una función de MINT. Por ejemplo, la función lstat() es redirecciónada a la función mint\_lstat de MINT.

Hay algunas llamadas al sistema que el emulador no soporta, por ejemplo fork() o sproc(), debido a que necesitan interactuar con el sistema operativo porque no está muy claro como el nuevo proceso será planificado. Libapp es la carpeta que contiene las interfaces necesarias para encargarse de este tipo de funciones.

La otra responsabilidad de libapp es emular Pthreads (especificada por IEEE POSIX 1003.1c). Cuando se compilan aplicaciones para SESC, el desarrollador debe elegir entre compilar una aplicación para correr bajo SESC o nativamente en el host. Finalmente libapp es también responsable por la API de locking.

#### 9.1.3.5.1 Thread API

- void sesc\_init(): Inicializa la librería y arranca el planificador interno de thread y transforma la única ejecución de proceso actual en un thread. Debe ser la primer llamada a una función de la API en una aplicación.
- sesc\_spawn(void (\*start routine) (void \*),void \*arg,long flags): Crea un nuevo hilo de control que se ejecuta concurrentemente con el proceso desde el cual se ejecuta (hilo principal). El nuevo hilo ejecutará start routine con los argumentos pasados como arg, las banderas aplicables son para definir sobre qué procesador se configurará la afinidad del hilo.
- void sesc\_wait(): bloquea el hilo hasta que uno de sus hijos termine su ejecución. Si el hilo no tiene hijos se reasume automáticamente. Esta función no retorna el pid del hilo como las funciones wait() tradicionales.
- void sesc\_self(): retorna el ID del hilo.
- int sesc\_suspend(int pid): suspende la ejecución del hilo con el pid pasado como argumento. El hilo es removido del loop de ejecución de instrucciones.
- int sesc\_resume(int pid): pasa el hilo suspendido a la cola de ejecución.
- int sesc\_yield(int pid): Explícitamente deja el control de ejecución para el hilo cuyo pid se pasa como parámetro. Si pid = -1, cualquier hilo puede ser enviado para ejecución.
- void sesc\_exit(): termina la ejecución del hilo actual.

#### 9.1.3.5.2 Sincronización API

- void sesc\_lock\_init(slock t \*lock): inicializa el bloqueo lock.
- void sesc\_lock(slock t \*lock): toma el bloqueo de lock.
- void sesc\_unlock(slock t \*lock): libera el bloqueo sobre lock.
- void sesc\_sema\_init(ssema t \*sema, int initialValue): inicializa un semáforo sema. initialValue es la cantidad de recursos que contiene el semáforo.
- void sesc\_psema(ssema t \*sema): emite un signal() sobre sema.
- void sesc\_vsema(ssema t \*sema): emite un wait() sobre sema.

## 10 ANEXO II

---

### 10.1 Instalación del Simulador SESC

#### 10.1.1 Instalar gcc 3.4

¿Porque necesitamos una versión anterior de gcc? Porque la versión más reciente causa errores al compilar el simulador, esto se debe a que SESC fue creado en 2005 usando gcc 3.4.

1. Descargar los siguientes paquetes desde gcc-3.4 :

1. [cpp-3.4\\_3.4.6-1ubuntu2\\_i386.deb](#)
2. [gcc-3.4\\_3.4.6-1ubuntu2\\_i386.deb](#)
3. [gcc-3.4-base\\_3.4.6-1ubuntu2\\_i386.deb](#)

Nota: Los números de versión de los archivos pueden no ser los mismos, pero no hay inconvenientes con eso. Por favor descargue solamente esos archivos.

2. Luego creamos un repositorio de los archivos descargados en el paso anterior, para más información referirse a <http://www.kubuntu-es.org/wiki/sistema/crear-repositorio-local-archivos-descargados>

1. Crear la carpeta donde estarán los archivos del paso 1 para crear el repositorio local.

```
: ~$ mkdir gcc-3.4
```

2. Ahora se creará el índice de repositorios

```
: ~$ dpkg-scanpackages gcc-3.4 /dev/null | gzip > gcc-3.4/Packages.gz
```

El comando **dpkg-scanpackages** es leer todos los archivos **.deb** que contiene el directorio y con **gzip** se crea el archivo **Packages.gz** que indica a apt cuáles son los paquetes que luego podremos instalar.

Si este paso no funciona instalar los paquetes de forma manual con el comando **sudo dpkg -i filename.deb**.

3. Ahora debemos agregar el repositorio al archivo source.list

```
: ~$ sudo gedit /etc/apt/source.list
```

Una vez abierto el archivo agregamos la siguiente línea al final

```
deb file:/home/$USER/ gcc-3.4/
```

4. Luego actualizamos la lista de repositorios e instalamos

```
: ~$ sudo apt-get update  
: ~$ sudo apt-get install gcc-3.4
```

3. Ahora debemos instalar g++ a través de dpkg.

1. Descargar los siguientes paquetes desde [gcc-3.4](#) :

- i. [g++-3.4\\_3.4.6-1ubuntu2\\_i386.deb](#)
  - ii. [libstdc++6-dev\\_3.4.6-1ubuntu2\\_i386.deb](#)

2. Y ejecutamos la instalación.

```
: ~$ sudo dpkg -i g++-3.4_3.4.6-1ubuntu2_i386.deb libstdc++6-dev_3.4.6-1ubuntu2_i386.deb
```

4. En este paso ya tenemos instalado gcc-3.4 en nuestro sistema, junto con la versión por defecto que es la 4.4. Esto podemos verlo listando las versiones de gcc:

```
: ~$ ls /usr/bin/gcc* -l
```

5. Agregar las opciones para las dos alternativas a gcc en nuestro sistema.

```
: ~$ sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.4 40  
: ~$ sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-3.4 30
```

6. Alternamos la versión de gcc a la 3.4

```
: ~$ sudo update-alternatives --config gcc
```

7. Una vez que seleccionamos la versión 3.4 ya tenemos funcionando la antigua versión de gcc. Podemos controlarlo usando:

```
: ~$ gcc -v
```

Lo que nos mostrará la versión de gcc que esta seleccionada para compilar por defecto.

### 10.1.2 Instalar Bison 2.3

1. Descargar Bison

```
: ~$ wget http://ftp.gnu.org/gnu/bison/bison-2.3.tar.gz
```

2. Extraer los archivos para instalar

```
: ~$ tar -xvzf bison-2.3.tar.gz
```

3. Si no tenemos instalado m4, instalarlo antes de pasar al paso 4.

```
: ~$ sudo apt-get install m4
```

4. Ahora entramos al directorio donde descomprimimos Bison y lo configuramos

```
: ~$ cd bison-2.3  
: ~$ ./configure --prefix=/usr/local/bison --with-libiconv-prefix=/usr/local/libiconv/
```

Nota: prefix indica el directorio donde se instalará bison.

5. Compilamos e instalamos

```
: ~$ make  
: ~$ sudo make install
```

6. Comprobamos que Bison este cargado en el PATH

```
: ~$ bison -v
```

Si el comando no fue encontrado debemos realizar el paso 7 para incluir bison al PATH.

7. Agregar la ruta al PATH

```
: ~$ export PATH=/usr/local/bison/bin/:$PATH
```

### 10.1.3 Generar SESC Utils

Actualmente, sescutils es una herramienta de compilación cruzada, la cual es usada para compilar programas en x86 para correr en MIPS. Es necesario ya que SESC es un simulador de arquitectura MIPS.

1. Descargar sescutils desde [sourceforge](#).
2. Mover el paquete a \$HOME/ para seguir correctamente este tutorial.
3. Descomprimir el paquete

```
: ~$ tar jxvf sescutils.tar.bz2
```

4. Los shell script de SESC están escritos en bash y como Ubuntu utiliza dash y no bash, debemos cambiar a este último.

```
: ~$ sudo rm /bin/sh  
: ~$ sudo ln -s /bin/bash /bin/sh
```

5. Modificar el archivo build-common en /home/\$USER/sescutils/build-mipseb-linux.

```
: ~$ gedit sescutils/build-mipseb-linux/build-common  
-----  
GNUSRC = $HOME/sescutils/src  
PREFIX = $HOME/sescutils/install
```

Nota: GNUSRC es la posición del código fuente. PREFIX es la posición donde los archivos binarios van a ser instalados.

6. Ingresamos a la carpeta build-mipseb-linux

```
: ~$ cd sescutils/build-mipseb-linux
```

7. Instalamos **build-1-binutils**. Acá pueden ocurrir 2 tipos de errores, esto se deben a que bison o flex no estén instalados.

```
: ~$ ./build-1-binutils
```

8. Instalamos **build-2-gcc-core**. Los errores en esta etapa se deben a que la versión de gcc utilizada para compilar no sea la 3.4 o que la versión de bison instalada sea la 2.4.

```
: ~$ ./build-2-gcc-core
```

9. Instalar **build-3-glibc**. El error en este paso es debido a un valor incorrecto de la variable BUILD en el archivo build-common. El comando *uname -p* no puede obtener la versión correcta del sistema. Para corregirlo debemos reemplazar el valor de BUILD a *BUILD="i686-pc-linux-gnu"* (o el valor correspondiente a su sistema).

```
: ~$ ./build-3-glibc
```

10. Instalar **build-4-gcc**.

```
: ~$ ./build-4-gcc
```

11. Instalar **build-5-gdb**. El error en este paso corresponde a una librería faltante, instalar libncurses5-dev con **sudo apt-get install libncurses5-dev**

```
: ~$ ./build-5-gdb
```

Ahora ya tenemos sescutils completamente generado. Usted puede utilizarlo para crear programas que correrá en SESC posteriormente.

### 10.1.4 Generar código fuente SESC

1. Instalar cvs

```
: ~$ sudo apt-get install cvs
```

2. Descargar código fuente de SESC por CVS

```
: ~$ cvs -d:pserver:anonymous@sesc.cvs.sourceforge.net:/cvsroot/sesc login  
: ~$ cvs -z3 -d:pserver:anonymous@sesc.cvs.sourceforge.net:/cvsroot/sesc co -P sesc
```

Nota: presionar Enter cuando se pide el password al loguearse.

3. Mover el código fuente descargado a /\$HOME/esesc

4. Crear directorio donde se generará el código.

```
: ~$ mkdir sesc-build  
: ~$ cd sesc-build
```

5. Configuramos el instalador

```
: ~$ ../esesc/configure
```

6. Instalar **binutils**

```
: ~$ sudo apt-get install binutils
```

7. Generamos

```
: ~$ make
```

ERROR: USHRT\_CHAR undefined

SOLUTION: incluir la cabecera limits.h en esesc/src/libcore/FetchEngine.cpp.

Agregar

**#include <limits.h>**

ERROR: /home/manio/SESC/build/../esesc/src/libmint/subst.cpp:52:26: error: linux/dirent.h: No such file or directory.

SOLUTION: No incluir linux/dirent.h, utilizar ***dirent.h***.

ERROR: 'uint32\_t' was not declared in this scope.

SOLUTION: agregar **#include <stdint.h>** en cualquier archivo que aparezca el error.

ERROR: "/usr/bin/ld: cannot find -lz"

SOLUTION: falta instalar paquete zlib1g-dev. **sudo apt-get install zlib1g-dev**.

ERROR: 'INT\_MAX' was not declared in this scope

SOLUTION: agregar **#include <limits.h>** en cualquier archivo que aparezca el error.

8. Una vez solucionado todos los errores y que la generación sea correcta estamos en condiciones de testear la instalación, para ello ejecutamos:

```
: ~$ make testsim
```

Como salida debemos obtener:

```
lac@lac-laptop:~/SESC/build$ make testsim
make[1]: `sesc.mem' is up to date.
Generating sesc.conf from: /home/manio/SESC/build/../esesc/confs/mem.conf
cp /home/lac/SESC/build/../esesc/confs/mem.conf sesc.conf
cp /home/lac/SESC/build/../esesc/confs/shared.conf .
./sesc.mem -h0x800000 -csesc.conf /home/ lac/SESC/build/../esesc/tests/crafty <
/home/lac/SESC/build/../esesc/tests/tt.in
static[0x1008db40-0x101b3dd4]      heap[0x101b4000-0x109b4000]      stack[0x109b4000-
0x111ac000] -> [0x41000000-0x4211e4c0]
Crafty v14.3
sesc_simulation_mark 0 (simulated) @30176641
White(1): sesc_simulation_mark 1 (simulated) @30225840
White(1): pondering disabled.
sesc_simulation_mark 2 (simulated) @30299339
White(1): noise level set to 0.
sesc_simulation_mark 3 (simulated) @30321649
White(1): search time set to 99999.00.
sesc_simulation_mark 4 (simulated) @30423769
White(1): verbosity set to 5.
sesc_simulation_mark 5 (simulated) @30449198
White(1): sesc_simulation_mark 6 (simulated) @30751014
White(1): search depth set to 2.
sesc_simulation_mark 7 (simulated) @30767155
White(1):
clearing hash tables
depth    time   score   variation (1)
1    ##.##  -0.67   axb5  c6xb5
1    ##.##  -0.08   a4a5
sesc simulation mark 8 (simulated) @33091197
```

La instalación se basó en la guía de <http://manio.org/category/computer-science/sesc-tutorial>.

### 10.1.4.1 Configurar SESC para instalar

```

Usage: ./configure [OPTION]... [VAR=VALUE]...
Para asignar variables de entorno (ejemplo: CC, CFLAGS, etc), especificarlas como VARIABLE=VALOR.
Las opciones por defecto son especificadas en corchetes.
Configuration:
  --cache-file=FILE cache test results in FILE [disabled]
  -C, --config-cache      alias for '--cache-file=config.cache'
  -n, --no-create         do not create output files
  --srcdir=DIR            find the sources in DIR [configure dir or
  '..']

Installation directories:
  --prefix=PREFIX          install architecture-independent files in
  PREFIX
  --exec-prefix=EPREFIX    install architecture-dependent files in
  EPREFIX

  --bindir=DIR             user executables [EPREFIX/bin]
  --sbindir=DIR            system admin executables [EPREFIX/sbin]
  --libexecdir=DIR          program executables [EPREFIX/libexec]
  --datadir=DIR            read-only architecture-independent data
  [PREFIX/share]
  --sysconfdir=DIR          read-only single-machine data [PRE-
  FIX/etc]
  --sharedstatedir=DIR     modifiable architecture-independent data
  [PREFIX/com]
  --localstatedir=DIR      modifiable single-machine data [PRE-
  FIX/var]
  --libdir=DIR              object code libraries [EPREFIX/lib]
  --includedir=DIR          C header files [PREFIX/include]
  --oldincludedir=DIR        C header files for non-gcc [/usr/include]
  --infodir=DIR              info documentation [PREFIX/info]
  --mandir=DIR              man documentation [PREFIX/man]

Opcional:
  --disable-FEATURE do not include FEATURE (same as --enable-
  FEATURE=no)
  --enable-FEATURE[=ARG]   include FEATURE [ARG=yes]
  --enable-taskscalar     Enable TaskScalar TLS support (default
  is no)
  --enable-power           Enable power model (default is no)
  --enable-gatherm         Enable GATHERM model (default is no)
  --enable-therm           Enable therm model (sescspot only) (de-
  fault is no)
  --enable-sesctherm       Enable SESCTHERM model (default is no)
  --enable-mispath         Enable simulation of wrong-path instruc-
  tions (default is no)
  --enable-debug            Enables assertions and LOG messages (de-
  fault is no)

```

```
--enable-debug-silent      In debug mode, do not show LOG messages  
(default is no)  
--enable-vmem-directory   Enables traffic filtering in VMEM (de-  
fault is no)  
--enable-vmem              Enables versioning memory subsystem mod-  
eling (default is no)  
--enable-ts-prof           Enables TaskScalar task profiling (de-  
fault is no)  
--enable-ts-risk-load-prof    Enables TaskScalar Risk-loads pro-  
filing (default is no)  
--enable-ts-nomlast        Enables NO_MERGELAST in TaskScalar (de-  
fault is no)  
--enable-ts-nomnext        Enables NO_MERGENEXT in TaskScalar (de-  
fault is no)  
--enable-smp                Enables SMP System (default is no)  
--enable-smpdebug          Enables SMP System advanced debug  
features (default is no)  
--enable-baad               Enables Bandwidth analyzer performance  
statistics (default is no)  
--enable-inorder            Enables optimized inorder pipeline (de-  
fault is no)  
--enable-condor              Enables linking sesc with condor librar-  
ies. Needed to run with standard universe. (default is no)  
--enable-trace               Enables trace-driven simulation (default  
is no)  
--enable-rsttrace            Enables RST trace-driven simulation (default is  
no)  
--enable-qemu                Enables qemu-driven simulation (default  
is no)  
--enable-mipsemul           Use new MIPs emulation in libemul (default is  
no)  
--enable-32bit               Build 32-bit simulator even on 64-bit  
host (default is no)
```

### 10.1.5 Pasos especiales para Ubuntu 10.10 de 64 bits

En un sistema operativo de 64 bits debemos compilar sescutils con las librerías de gcc para 64 bits y el simulador con las de 32 bits.

La única salvedad que hay que hacer para la generación de sescutils es que los paquetes de gcc y g++ descargados deben ser los que tienen como sufijo *amd64*.

Para compilar el generador en 32 bits se necesita instalar algunos paquetes más:

- lib32ncurses5-dev
- g++-multilib
- lib32z1-dev

Estos paquetes los instalamos a través de comando *apt-get install* con permisos de root.

Una vez que tenemos instalados los paquetes anteriores vamos a modificar unas líneas al archivo **Makefile.defs**, que se encuentra dentro de */esesc/src/*. Debemos buscar las líneas donde están declaradas **CFLAGS** y **LDFLAGS** y le debemos agregar el parámetro **-m32**.

```
CFLAGS = $(ABI) $(COPTS) $(INC) $(PDEFS) $(DEFS) -m32
LDFLAGS = $(ABI) $(LOPTS) -m32
```

## 10.2 Compilar programa para simular con SESC

Debemos exportar sescutils y SESC al PATH:

```
: ~$ export PATH=$HOME/sescutils/install/bin:$HOME/sesc-build:$PATH
```

Ejemplo de prueba HolaSESC:

```
#include <stdio.h>

int
{
    printf("hello sesc\n");
    return
}
```

Compilamos el programa con **mipseb-linux-gcc**:

```
: ~/sescworkdir$ mipseb-linux-gcc -mips2 -mabi=32 -static -Wa,-non_shared
-mno-abicalls
-Wl,-script=$HOME/sescutils/install/mipseb-
linux/lib/ldscripts/mint.x,-static hellosesc.c -o hellosesc.out
```

### **-O2 -mips2 -mabi=32 -fno-PIC -mno-abicalls**

Debemos compilar con las mismas opciones que *glibc* (los parámetros utilizados en la generación en *build-3-glibc* de sescutils)

### **-Wl,option**

Con este parámetro podemos pasar opciones al linkeador. Se pueden setear varias opciones separadas por comas.

### **-script=scriptfile**

Pasamos en **scriptfile** la ruta al fichero que vamos a utilizar como script del linkeador.

### **-static**

No linkea nuevamente las librerías compartidas.

Nota: Al compilar el programa debemos estar seguros que la versión de gcc seleccionada en nuestro sistema sea la 3.4.

Una vez que tenemos compilado correctamente el archivo lo corremos en el simulador

```
:~/sescworkdir$ sesc.mem -c../esesc/confs/mem.conf ./hellosesc.out < ../esesc/tests/tt.in
```

### 10.2.1 Programas multi-thread

1. Incluir `#include <stdint.h>` en el archivo ***sescapi.h*** que se encuentra en la carpeta `$HOME/esesc/src/libapp`, verificar que no se haga el include dentro de un `#ifdef`.
2. Crear el objeto de `sesc_thread.c`

```
: ~/esesc/src/libapp$ mipseb-linux-gcc -g -mips2 -mabi=32 -Wa,-non_shared -mno-abicalls -c -o sesc_thread.o sesc_thread.c
```

3. Crear el ejecutable. Notar que se linkea `mint.x`, el cual es elf para sesc mips.

```
: ~/workdir$ mipseb-linux-gcc -o hello.sesc hello.c -g -mips2 -mabi=32 -Wa,-non_shared -mno-abicalls /$HOME/esesc/src/libapp/sesc_thread.o -static -Wl,--script=/$HOME/sescutils/install/mipseb-linux/lib/ldscripts/mint.x,-static
```

4. Correr el simulador.

```
: ~/workdir$ ../sesc-build/sesc.smp -c ../esesc/confs/smp.conf hello.sesc
```

5. Mostrar el reporte del programa simulado

```
: ~/workdir$ ../esesc/scripts/report.pl -last
```

## 10.3 Incorporar el simulador a la IDE KDevelop

### 10.3.1 Instalación de IDE KDevelop3

El fin que tiene este apartado del documento es explicar cómo instalar KDevelop 3.5 en Ubuntu 10.10 para poder emplearla a la hora de modificar el simulador, logrando así una manera de trabajar más sencilla. Básicamente debemos seguir los siguientes pasos:

1. Eliminar cualquier instalación previa de kdevelop

```
: $sudo apt-get remove --purge kdevelop*
```

2. Agregar los repositorios de los que descargaremos kdevelop con kde3

```
: $sudo nano /etc/apt/sources.list
```

y añadimos:

```
deb http://ppa.launchpad.net/kde3-maintainers/ppa/ubuntu lucid main
deb-src http://ppa.launchpad.net/kde3-maintainers/ppa/ubuntu lucid main
```

3. Añadir la clave pública GPG del repositorio

```
$wget http://apt.pearsoncomputing.net/public.gpg
$sudo apt-key add public.gpg
```

4. Actualizar la lista de paquetes disponibles en los repositorios  

```
$sudo apt-get update
```
5. Instalar la última versión de kdevelop con kde3 junto a la documentación.  

```
$sudo apt-get install kdevelop-kde3 kdevelop-kde3-doc
```
6. Instalar el emulador de terminal para kde.  

```
$sudo apt-get install konsole
```
7. Eliminar cualquier instalación previa de la librería libtool, ya que es incompatible la versión de libtool que hay para Ubuntu con la que necesita kdevelop.  

```
$sudo apt-get remove libtool
```
8. Descargar e instalar la versión 1.5.26-1 de libtool (*amd64 en nuestro caso*).  

```
Para i386
$wget http://mirrors.kernel.org/ubuntu/pool/main/libt/libtool/libtool_1.5.26-1ubunt...
$sudo dpkg -i libtool_1.5.26-1ubuntu1_amd64.deb
$rm libtool_1.5.26-1ubuntu1_amd64.deb
Para amd64
$wget http://mirrors.kernel.org/ubuntu/pool/main/libt/libtool/libtool_1.5.26-1ubunt...
$sudo dpkg -i libtool_1.5.26-1ubuntu1_i386.deb
```
9. Instalar build-essential  

```
$sudo apt-get install build-essential
```
10. Ejecutar kdevelop  

```
$/opt/kde3/bin/kdevelop
```

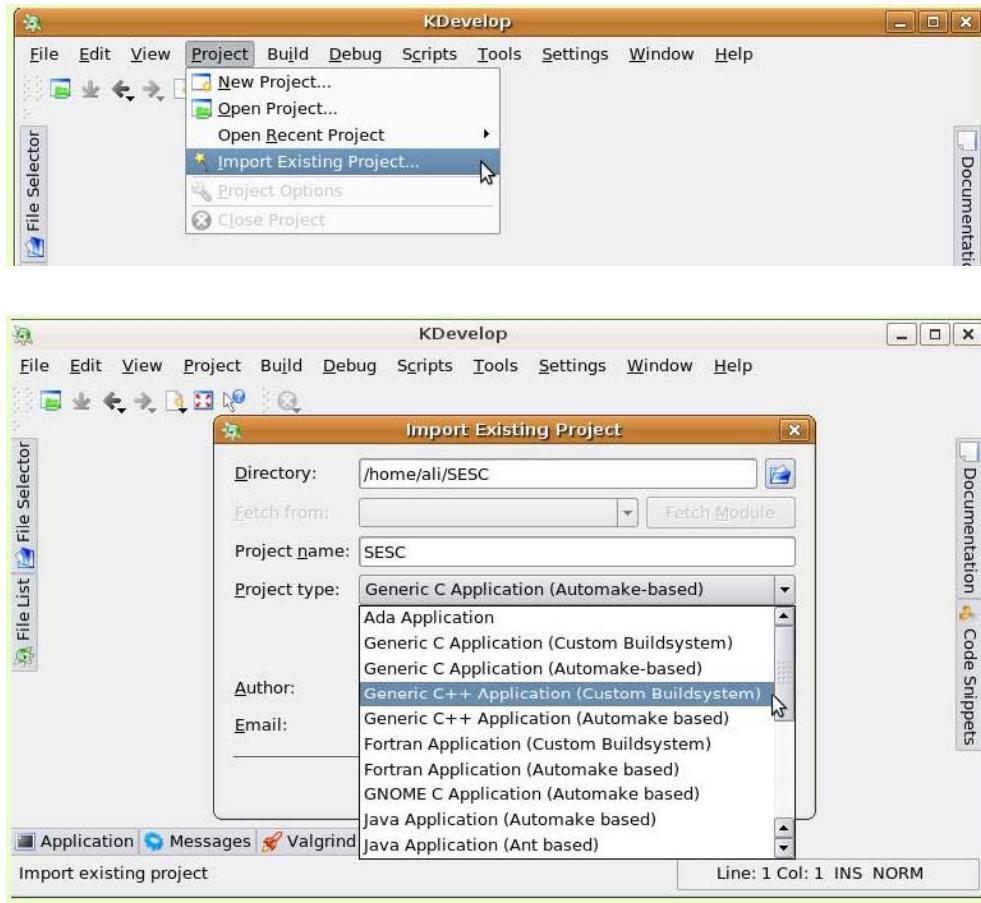
### 10.3.2 Como Integrar el simulador SESC en KDevelop3 (18)

Antes de integrar SESC a la IDE es recomendable instalar la herramienta de documentación Doxygen.

***\$sudo apt-get install doxygen***

Una vez concluida la instalación pasamos a integrar el proyecto.

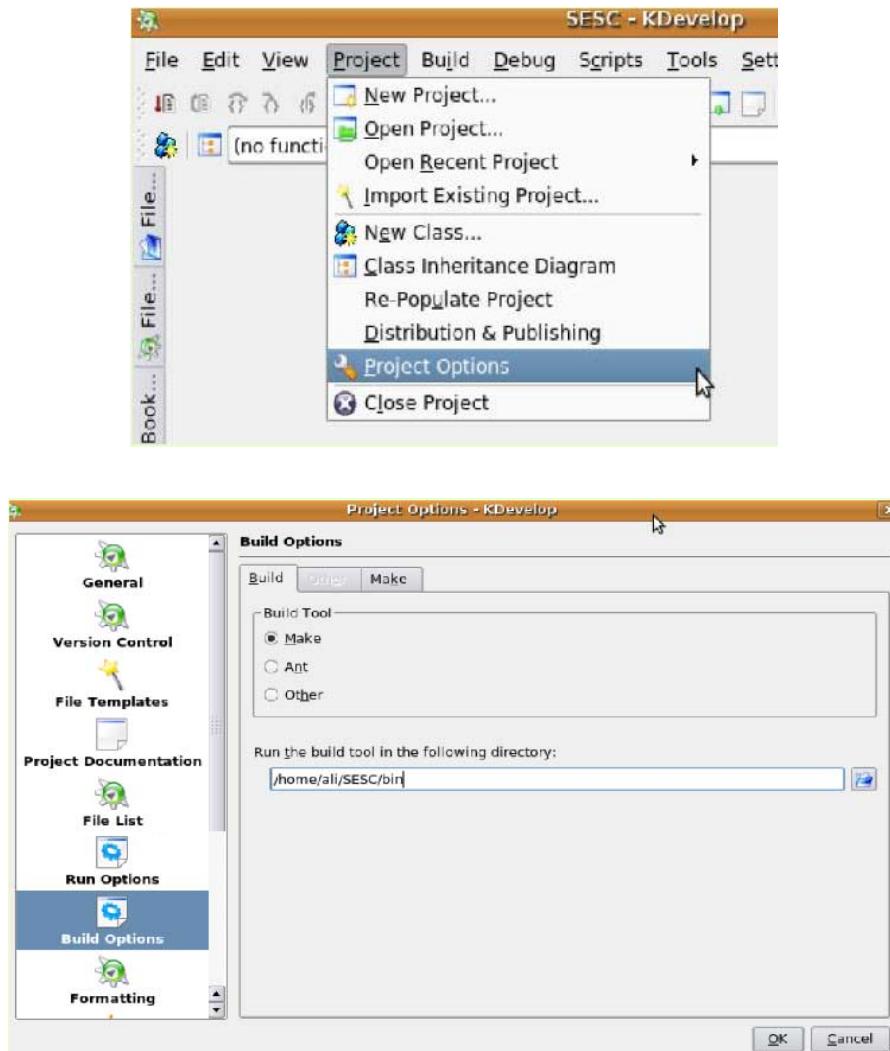
1. Importar el proyecto



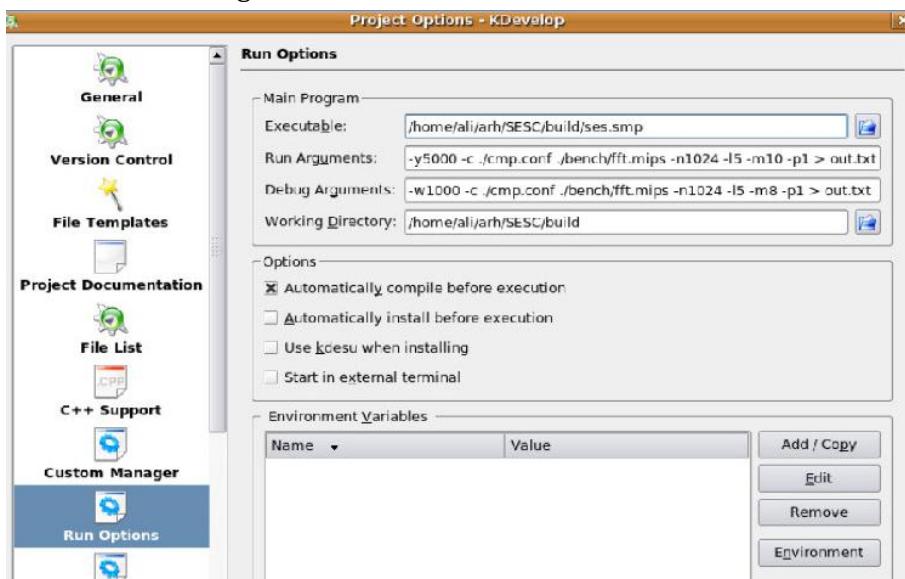
2. Then you need to populate your Project file with all of the source codes in the SESC directory.
3. After choosing the suitable filetypes (.h .cpp .c and Makefile) you need to select the sources.



4. Ahora debemos configurar una serie de parámetros, Build Options, se debe setear correctamente el PATH del Makefile y de los archivos binarios.



5. Luego debemos modificar las opciones de Run Options, en donde debemos setear el archivo de Ejecución (sesc.smp) y si es necesario colocar las opciones correspondientes de debug.



## 10.4 Mapeo de instrucciones MIPS-MINT

MIPS	MINT	MIPS	MINT
Ulimit	mint_ulimit	Recv	mint_recv
Excel	mint_execl	recvfrom	mint_recvfrom
Excel	mint_execle	recvmsg	mint_recvmsg
Execlp	mint_execlp	getsockopt	mint_getsockopt
Execv	mint_execv	setsockopt	mint_setsockopt
Execve	mint_execve	Select	mint_select
Execvp	mint_execvp	cachectl	mint_null_func
Mmap	mint_mmap	oserror	mint_oserror
open64	mint_open	setoserror	mint_setoserror
Open	mint_open	Perror	mint_perror
Close	mint_close	Times	mint_times
Read	mint_read	Getdtablesize	mint_getdtablesize
write	mint_write	Syssgi	mint_syssgi
readv	mint_readv	Time	mint_time
writev	mint_writev	Munmap	mint_munmap
creat	mint_creat	Malloc	mint_malloc
link	mint_link	Calloc	mint_calloc
unlink	mint_unlink	Free	mint_free
rename	mint_rename	Cfree	mint_free
chdir	mint_chdir	realloc	mint_realloc
chmod	mint_chmod	Getpid	mint_getpid
fchmod	mint_fchmod	getppid	mint_getppid
chown	mint_chown	Clock	mint_clock
fchown	mint_fchown	gettimeofday	mint_gettimeofday
lseek64	mint_lseek64	Sysmp	mint_sysmp
lseek	mint_lseek	getdents	mint_getdents
access	mint_access	getdents	mint_getdents
stat64	mint_stat64	sproc	mint_notimplemented
lstat64	mint_lstat64	sprocs	mint_notimplemented
fstat64	mint_fstat64	m_fork	mint_notimplemented
xstat64	mint_xstat64	m_next	mint_notimplemented
fxstat64	mint_fxstat64	m_set_procs	mint_notimplemented
stat	mint_stat	m_get_numprocs	mint_notimplemented
lstat	mint_lstat	m_get_myid	mint_notimplemented
fstat	mint_fstat	m_kill_procs	mint_notimplemented
xstat	mint_xstat	m_parc_procs	mint_notimplemented
dup	mint_dup	m_rele_procs	mint_notimplemented
pipe	mint_pipe	m_sync	mint_notimplemented
symlink	mint_symlink	m_lock	mint_notimplemented
readlink	mint_readlink	m_unlock	mint_notimplemented
umask	mint_umask	fork	mint_notimplemented
getuid	mint_getuid	Wait	mint_notimplemented

geteuid	mint_geteuid	wait3	mint_notimplemented
getgid	mint_getgid	Waitpid	mint_notimplemented
getegid	mint_getegid	pthread_create	mint_notimplemented
gethostname	mint_gethostname	pthread_lock	mint_notimplemented
getdomainname	mint_getdomainname	isatty	mint_isatty
setdomainname	mint_setdomainname	ioctl	mint_ioctl
Socket	mint_socket	prctl	mint_prctl
connect	mint_connect	fcntl164	mint_fcntl
Send	mint_send	fcntl	mint_fcntl
Sendto	mint_sendto	cerror64	mint_cerror
sendmsg	mint_sendmsg	cerror	mint_cerror



## 11 ANEXO III

---

### 11.1 Archivo de configuración Original

Se analizará el archivo de configuración de un microprocesador openSparcT1 (contenido en la carpeta confs del código fuente).

```
procsPerNode = 32      #cantidad de procesadores por nodo
cacheLineSize = 32      #tamaño del bus de cache
cpucore[0:$|(procsPerNode)-1] = 'issueX' #se define el tipo de procesador al cual pertenecen los
procesadores. Se puede definir cores heterogéneos.

#####
# Sistema          #
#####

enableICache = trae #habilita la cache de instrucciones
NoMigration = trae #no migra el hilo hacia otros procesadores una vez iniciado
tech      = 0.10
pageSize   = 4096 #tamaño de pagina en bytes
fetchPolicy = 'outorder' #politica de busqueda (outorder - inorder)
issueWrongPath = trae #habilita la ejecución de instrucciones en salto mal tomado.
technology = 'techParam'

#####
# Reloj          #
#####

[techParam]
clockTreeStyle = 1 # 1 para Htree, 2 para balHtree
tech      = 70 # nm
frequency = 1e9 # Hz
skewBudget = 20 # ps
areaOfChip = 200 # in mm^2
loadInClockNode = 20 # en pF
optimalNumberOfBuffer = 3
```

```
#####
# Procesador          #
#####

[issueX] #etiqueta que corresponde con cpucore definido
frequency    = 1e9
#smtContexts = 4
#smtFetches4Clk = 1
#smtDecodes4Clk = 1
#smtIssues4Clk = 1
areaFactor   = 0.1
inorder      = true #politica de expedicion
fetchWidth   = 2 #cant de instrucciones buscadas
instQueueSize = 2
issueWidth   = 1
retireWidth  = 1
decodeDelay  = 1
renameDelay  = 1
wakeupDelay  = 0 # -> 6+3+6+1+1=17 salto mal predicto
maxBranches  = 6
bb4Cycle     = 1
maxIRequests = 10 #maxima cantidad de inst. solicitadas
interClusterLat = 1
intraClusterLat = 1
cluster[0]    = 'clusterIssueX'
stForwardDelay = 3
maxLoads     = 4 #max cantidad de loads
maxStores    = 4 #maxima cantidad de stores
regFileDelay = 1
robSize      = 8  #Tamaño del buffer de reordenamiento
intRegs      = 32
fpRegs       = 32
bpred        = 'BPredIssueX'
dtlb         = 'FXDTLB'
itlb         = 'FXITLB'
dataSource   = "DMemory DL1"
instrSource  = "IMemory IL1"
enableICache = true
OSType       = 'dummy'
```

#### # Unidades funcionales enteras

```
[ClusterIssueX]
winSize  = 4
recycleAt = 'Execute'
```

```
schedNumPorts = 0
schedPortOccp = 0
wakeUpNumPorts= 0
wakeUpPortOccp= 0
wakeupDelay = 1
schedDelay = 0 # Minima latencia (como latencia intracluster)
iStoreLat = 1
iStoreUnit = 'LDSTIssueX'
iLoadLat = 1
iLoadUnit = 'LDSTIssueX'
iALULat = 1
iALUUnit = 'ALUIssueX'
iBJLat = 1
iBJUnit = 'ALUIssueX'
iDivLat = 72
iDivUnit = 'ALUIssueX'
iMultLat = 7
iMultUnit = 'ALUIssueX'
fpALULat = 26
fpALUUnit = 'FPIssueX'
fpMultLat = 29
fpMultUnit = 'FPIssueX'
fpDivLat = 54
fpDivUnit = 'FPIssueX'
```

**[FPIssueX]**

```
Num = 1
Occ = 3
```

**[LDSTIssueX]**

```
Num = 1
Occ = 1
```

**[ALUIssueX]**

```
Num = 1
Occ = 1
```

**# mecanismo de predicción de salto**

```
[BPredIssueX]
type      = "static"
BTACDelay = 0
btbSize   = 128
btbBsize  = 1
btbAssoc  = 2
btbReplPolicy = 'LRU'
btbHistory = 0
```

```
rasSize = 4
```

**# mecanismo de traduccion de direcciones de memoria**

```
[FXDTLB]
```

```
size = 64*8
```

```
assoc = 4
```

```
bsize = 8
```

```
numPorts = 1
```

```
replPolicy = 'LRU'
```

```
[FXITLB]
```

```
size = 64*8
```

```
assoc = 4
```

```
bsize = 8
```

```
numPorts = 1
```

```
replPolicy = 'LRU'
```

```
#####
```

```
# Memoria #
```

```
#####
```

**# cache L1 de instrucciones**

```
[IMemory]
```

```
deviceType = 'icache'
```

```
size = 16*1024
```

```
assoc = 4
```

```
bsize = $(cacheLineSize)
```

```
writePolicy = 'WT'
```

```
replPolicy = 'random'
```

```
numPorts = 1
```

```
portOccp = 1
```

```
hitDelay = 2
```

```
missDelay = 1      #este valor se suma al delay por acierto (3) MSHR = "iMSHR"
```

```
lowerLevel = "L1L2Bus L1L2"
```

```
[iMSHR]
```

```
type = 'single'
```

```
size = 4
```

```
bsize = $(cacheLineSize)
```

**# cache L1 de datos**

```
[DMemory]
```

```
deviceType = 'cache'
```

```
size = 8*1024
```

```
assoc = 4
```

```
bsize = $(cacheLineSize)
```

```
writePolicy = 'WT'
replPolicy = 'random'
numPorts = 1
portOccp = 1
hitDelay = 3
missDelay = 3 #este valor es agregado el hitDelay
maxWrites = 8
MSHR = "DMSHR"
lowerLevel = "L1L2Bus L1L2"

[DMSHR]
type = 'single'
size = 8
bsize = $(cacheLineSize)

# bus between L1s and L2
[L1L2Bus]
deviceType = 'bus'
numPorts = 1
portOccp = 0 # assuming 256 bit bus
delay = 3
lowerLevel = "L2Cache L2 shared"

# private L2
[L2Cache]
deviceType = 'cache'
size = 2*1024*1024
assoc = 16
bsize = $(cacheLineSize)
writePolicy = 'WB'
replPolicy = 'LRU' #protocolos de reemplazo de cache
protocol = 'MESI' #protocolo de coherencia de cache
numPorts = 1 # uno para L1 y una para snooping
portOccp = 1
hitDelay = 10
missDelay = 10 # no se suma a delay por acierto
displNotify = false
MSHR = 'L2MSHR'
lowerLevel = "MemoryBus SysBus"

[L2MSHR]
size = 32
type = 'single'
bsize = $(cacheLineSize)

[SystemBus]
```

```
deviceType  = 'systembus'
numPorts    = 1
portOccp   = 1
delay      = 1
lowerLevel  = "MemoryBus MemoryBus"

[MemoryBus]
deviceType  = 'bus'
numPorts    = 1
portOccp   = $(cacheLineSize) / 4 # asumiendo 4 bytes/cycle
delay      = 1
lowerLevel  = "Memory Memory"

[Memory]
deviceType  = 'niceCache'
size       = 64
assoc      = 1
bsize      = 64
writePolicy = 'WB'
replPolicy  = 'LRU'
numPorts    = 1
portOccp   = 1
hitDelay   = 85
missDelay  = 85
MSHR       = NoMSHR
lowerLevel  = 'voidDevice'

[NoMSHR]
type = 'none'
size = 128
bsize = 64

[voidDevice]
deviceType  = 'void'
```

## 11.2 Archivo de configuración Modificado

Por cuestiones de Practicidad solo se han puesto las diferencias y modificaciones que sufrió el Archivo de Conf. Específicamente en la sección Memoria, ya que no tiene sentido alguno repetir toda la misma información.

```
#####
# Memoria      #
#####
```

---

```
# instruction source
[IMemory]
```

```
deviceType = 'icache'
size      = 16*1024
assoc     = 4
bsize     = $(cacheLineSize)
writePolicy = 'WT'
replPolicy = 'random'
numPorts   = 1
portOccp   = 1
hitDelay   = 2
missDelay  = 1      # this number is added to the hitDelay
MSHR      = "iMSHR"
lowerLevel  = "L1L2Bus L1L2"

[iMSHR]
type = 'single'
size = 4
bsize = $(cacheLineSize)

# data source
[DMemory]
deviceType = 'cache'
size      = 8*1024
assoc     = 4
bsize     = $(cacheLineSize)
writePolicy = 'WT'
replPolicy = 'random'
numPorts   = 1
portOccp   = 1
hitDelay   = 3
missDelay  = 3      #this number is added to the hitDelay
maxWrites  = 8
MSHR      = "DMSHR"
lowerLevel  = "L1L2Bus L1L2"

# data source
[PMemory]
#deviceType = 'cache'
deviceType = 'petriCache'
size      = 2*1024
assoc     = 4
bsize     = $(cacheLineSize)
writePolicy = 'WB'
replPolicy = 'LRU'
numPorts   = 1
portOccp   = 1
hitDelay   = 1
```

```
missDelay  = 3          #this number is added to the hitDelay
maxWrites  = 1
MSHR      = NoMSHR
lowerLevel  = "MemoryBus SysBus"
#lowerLevel  = "nada"

[DMSHR]
type = 'single'
size = 8
bsize = ${cacheLineSize}

#MSHR de petris
[PMSHR]
type = 'single'
size = 8
bsize = ${cacheLineSize}

# bus between L1s and L2
[L1L2Bus]
deviceType = 'bus'
numPorts  = 1
portOccp  = 0          # assuming 256 bit bus
delay     = 3
lowerLevel = "L2Cache L2 shared"

# private L2
[L2Cache]
deviceType  = 'cache'
size        = 2*1024*1024
assoc       = 16
bsize       = ${cacheLineSize}
writePolicy = 'WB'
replPolicy  = 'LRU'
protocol    = 'MESI'
numPorts    = 1          # one for L1, one for snooping
portOccp   = 1
hitDelay   = 10
missDelay  = 10          # exclusive, i.e., not added to hitDelay
displNotify = false
MSHR       = 'L2MSHR'
lowerLevel  = "MemoryBus SysBus"

[L2MSHR]
size = 32
type = 'single'
bsize = ${cacheLineSize}
```

```
[SystemBus]
deviceType = 'systembus'
numPorts = 1
portOccp = 1
delay = 1
lowerLevel = "MemoryBus MemoryBus"

[MemoryBus]
deviceType = 'bus'
numPorts = 1
portOccp = ${cacheLineSize} / 4 # assuming 4 bytes/cycle bw
delay = 1
lowerLevel = "Memory Memory"

[Memory]
deviceType = 'niceCache'
size = 64
assoc = 1
bsize = 64
writePolicy = 'WB'
replPolicy = 'LRU'
numPorts = 1
portOccp = 1
hitDelay = 85
missDelay = 85
MSHR = NoMSHR
lowerLevel = 'voidDevice'

[NoMSHR]
type = 'none'
size = 128
bsize = 64

[voidDevice]
deviceType = 'void'

[nada]
deviceType = 'void'
```



## 12 ANEXO IV

### 12.1 Interpretación de los resultados de una simulación

```

①# Bench : ../sesc-build/sesc.smp -c ../esesc/confs/smp.conf pipes-inicio.o
②# File : sesc_pipes-inicio.o.dkyVce : Thu May 12 01:43:17 2011
③ Exe Speed      Exe MHz      Exe Time      Sim Time (5000MHz)
    564.441 KIPS   3.1907 MHz   0.170 secs   0.108 msec
④ Proc Avg.Time BPType      Total      RAS      BPred      BTB      BTAC
    0 221.459 hybrid      90.42% ( 97.35% of 11.53%)  89.51% ( 96.45% of 53.06%)  0.00%
          nInst      BJ      Load      Store      INT      FP      : LD Forward,      Replay      : Worst
          0      95955  19.49%  19.27%  13.83%  47.41%  0.00%      :      36.47%  119 inst/repl      : LDSTI
Proc IPC      Cycles      Busy      LDQ      STQ      IWin      ROB      Regs      Ports      TLB      maxBr      MisBr      Br4Clk      Other
    0  0.18      542407  4.4      0.2      4.0      0.7      0.9      9.3      0.0      0.0      0.0      0.0      79.7      0.0      0.7
#####
⑤ Proc Cache Occ MissRate (RD, WR) %DMemAcc MB/s : ...
    0 DL1 0.0  1.68% ( 0.5%, 1.2%) 78.73%  0.12GB/s : L1L2 3853.56 MB/s : SysBus 0 MB/s : Memc
#####
⑦ Proc Cache Occ MissRate (RD, WR) %DMemAcc MB/s : ...
    0 IL1 0.0  3.71% ( 3.7%, 0.0%) 88.50%  0.31GB/s : L1L2 3853.56 MB/s : SysBus 0 MB/s : Memc

```

**Figura 35. Resultado de una simulación**

A continuación se detallan los datos que se obtienen luego de correr un programa en SESC, los números corresponden a los indicados en la Figura 35.

1. Nos indica la aplicación simulada, con los parámetros indicados al simulador.
2. Es el archivo de salida generado por el simulador, desde el cual se obtuvo el reporte del simulador.
3. *Exe Speed*: velocidad de ejecución en KIPS (Nro instrucciones / tiempo ejecución\*1000).

*Exe MHz*: frecuencia de ejecución en MHz (tiempo de ejecución / cantidad de ciclos).

*Exe Time*: tiempo de ejecución.

*Sim Time*: tiempo de simulación ((1e-3 / frecuencia) \* número de ciclos).

4. *Proc*: Indica el número del procesador.

*Avg.Time*: Tiempo promedio de salto.

*BPType*: Tipo de predictor de salto.

*Total*: Porcentaje de saltos acertados.

*RAS(Row Access Strobe)*: indica el porcentaje de aciertos en RAS y el RAS respecto al número de saltos.

- BPred*: Porcentaje de aciertos del predictor.
- BTB (Branch Target Buffer)*: Porcentaje de aciertos de la BTB y la cantidad de accesos a la BTB respecto a la cantidad de saltos.
- BTAC (Branch Target Address Cache)*: Porcentaje de accesos a la BTAC.
- nInst*: cantidad de instrucciones ejecutadas sobre el procesador.
- Bl*: porcentaje de instrucciones de saltos.
- Load*: porcentaje de instrucciones de load.
- Store*: porcentaje de instrucciones de store.
- INT*: porcentaje de instrucciones enteras.
- FP*: porcentaje de instrucciones de punto flotante.
- LD Forward*: porcentaje de forwarding respecto de la cantidad de instrucciones de load.
- Replay*: Cantidad de instrucciones repetidas???
- Worst Unit*: cantidad de clocks necesarios en la peor unidad funcional.
- IPC*: instrucciones por ciclo.
- Cycles*: cantidad de ciclos.
- Busy*: porcentaje de ciclos ocupado del procesador con respecto a la cantidad de ciclos.
- LDQ, STQ*: porcentaje de load o stores expedidas respecto de cantidad de instrucciones ideales que podrían ser expedidas.
- Iwin*: porcentaje de uso de la ventana de instrucciones.
- ROB*: porcentaje de uso del buffer de reordenamiento.
- Regs*: porcentaje de uso de los registros.
- Ports*: porcentaje de conflictos de puertos.
- TLB*: porcentaje de uso de la TLB.
- maxBR*: porcentaje de uso de saltos.
- misBR, BR4clk, Other*: solo sirven como control.
5. *Proc*: Indica el número del procesador.
- Cache*: Tipo de cache.
- Miss Rate (RD, RW)*: Tasa de errores, lectura y escritura.
- %DMemAccess*: porcentaje de accesos a memoria respecto de load más stores.
6. Los datos obtenidos son iguales a los del punto 5 pero respecto de la cache de instrucciones.

Para obtener otros datos de los logs generados por el simulador le podemos pasar los siguientes parámetros al script de reportes: **./report.pl [options] <sesDump>**

- ag** : Reports for all the stat files in current directory\n;
- last** : Reports the newest stat file in current directory\n;
- long** : Reports much more information (verbose)\n;
- [no]bpred** : Deactivae/Activate branch predictor statistics\n;
- [no]cpu** : Deactivae/Activate CPU statistics\n;
- [no]inst** : Deactivae/Activate instruction statistics\n;
- [no]sim** : Show results without comments\n;

```


|        |                                                           |
|--------|-----------------------------------------------------------|
| -table | : Statistics table sumary (god for scripts)\n";           |
| -baad  | : BAAD statistics\n";                                     |
| -cg    | : CriticalityGraph statistics\n";                         |
| -dead  | : List report files whose simulations didn't complete\n"; |


```

## 12.2 Códigos y Mediciones

Para mayor detalle de las instrucciones no propias del lenguaje C, consultar el Anexo I, apartado Análisis detallado del Simulador.

### 12.2.1 SINCRONIZACION

#### Código Empleado para medir la sincronización en el Simulador Original

```

#include "/home/user/esesc/src/libapp/sescapi.h"
#include "/home/user/esesc/src/libapp/sesc_locks.c"
#include "/home/user/esesc/src/libapp/sesc_events.c"
#include <stdio.h>
#include <stdlib.h>

#define NUM_THREADS 2
#define LOOP_ext 500000
#define TAM_ARRAY 2

int variable_compartida[TAM_ARRAY];

ssema_t Semaforo;
ssema_t *semaA=&Semaforo;
ssema_t *semaB=&Semaforo;

void work(void *threadid)

{
    int tid= sesc_self();
    int i=0,k=0;
    int *p;

    if(tid==1) {

        for(i=0;i<LOOP_ext;i++) {

            sesc_vsema(semaA);           //toma semáforo A

            for(k=0;k<TAM_ARRAY;k++) {
                variable_compartida[k]=i;
            }

            sesc_psema(semaB);          //libera semáforo B
        }
    } else if(tid==2) {

```

```

        for(i=0;i<LOOP_ext;i++) {

            sesc_fast_sim_begin_();
            for(int j=0;j<10;j++) int r = 22;
            sesc_fast_sim_end_();

            sesc_vsema(semaB); //toma semáforo B

        for(k=0;k<TAM_ARRAY;k++) {
            variable_compartida[k]=i;
        }

            sesc_psema(semaA); //libera semáforo A
        }

        sesc_exit(0);
    }
int main()

{
    int t;
    sesc_init();
    sesc_sema_init(semaA , (int32_t) 1);
    sesc_sema_init(semaB , (int32_t) 0);

    for(t=0;t<NUM_THREADS;t++) {

        sesc_spawn(work, (void *) t, SESC_FLAG_NOMIGRATE|SESC_FLAG_MAP|t+1);

    }

    sesc_exit(0);
}

// Las mediciones fueron llevadas a cabo variando el valor de LOOP_ext y TAM_ARRAY

```

#### Código Empleado para medir la sincronización en el Simulador Modificado

```

#include "/home/user/esesc/src/libapp/sescapi.h"
#include "/home/user/esesc/src/libapp/sesc_locks.c"
#include "/home/user/esesc/src/libapp/sesc_events.c"
#include <stdio.h>
#include <stdlib.h>

#define NUM_THREADS 2
#define LOOP_ext 500000
#define TAM_ARRAY 2

int variable_compartida[TAM_ARRAY];

void work(void *threadid)
{

```

```

int tid= sesc_self();
int i=0,k=0;
int *p;

if(tid==1) {

    for(i=0;i<LOOP_ext;i++) {

        p = (int *) 405000000;
        *p=25;

        for(k=0;k<TAM_ARRAY;k++) {
            variable_compartida[k]=i;
        }

        p = (int *) 405000001;
        *p=25;
    }

} else if(tid==2) {

    for(i=0;i<LOOP_ext;i++) {
        sesc_fast_sim_begin_();
        for(int j=0;j<10;j++) int r = 22;
        sesc_fast_sim_end_();

        p = (int *) 405000002;
        *p=25;

        for(k=0;k<TAM_ARRAY;k++) {
            variable_compartida[k]=i;
        }

        p = (int *) 405000003;
        *p=25;
    }
}

sesc_exit(0);
}

int main()
{
    int t;
    for(t=0;t<NUM_THREADS;t++) {
        sesc_spawn(work, (void *) t, SESC_FLAG_NOMIGRATE|SESC_FLAG_MAP|t+1);
    }
    sesc_exit(0);
}

// Las mediciones fueron llevadas a cabo variando el valor de LOOP_ext y TAM_ARRAY

```

## 12.2.2 ESCRITOR/ESCRITOR

### ESCRITOR/ESCRITOR (para correr con simulador original)

```
#include "$HOME/esesc/src/libapp/sescapi.h"
#include "$HOME/esesc/src/libapp/sesc_locks.c"
```

```
#include "$HOME/esesc/src/libapp/sesc_events.c"

#define NUM_THREADS 2
#define LOOP_ext 200
#define TAM_ARRAY 204800/LOOP_ext

int variable_compartida[TAM_ARRAY];

ssema_t semaA;
ssema_t semaB;

void work(void *threadid)
{
    int tid= sesc_self();

    int i=0,k=0;

    //int *p;

    if(tid==1) {

        for(i=0;i<LOOP_ext;i++) {

            //toma semaforo A
            sesc_psema(&semaA);

            for(k=0;k<TAM_ARRAY;k++) {
                variable_compartida[k]=i;
            }

            //libera semaforo B
            sesc_vsema(&semaB);
        }
    }

    } else if(tid==2) {

        for(i=0;i<LOOP_ext;i++) {

            //toma semaforo B
            sesc_psema(&semaB);

            for(k=0;k<TAM_ARRAY;k++) {
                variable_compartida[k]=i;
            }

            //libera semaforo A
            sesc_vsema(&semaA);
    }
}
```

```

        }
    }

    sesc_exit(0);
}

int main()
{
    int t;

    sesc_init();

    sesc_sema_init(&semaA, 1);
    sesc_sema_init(&semaB, 0);

    for(t=0;t<NUM_THREADS;t++) {
        sesc_spawn(work, (void *) t,
SESC_FLAG_NOMIGRATE|SESC_FLAG_MAP|t+1);
    }
    sesc_exit(0);
}

```

**ESCRITOR/ESCRITOR (para correr con simulador que implementa Petri)**

```

#include "$HOME/esesc/src/libapp/sescapi.h"
#include "$HOME/esesc/src/libapp/sesc_locks.c"
#include "$HOME/esesc/src/libapp/sesc_events.c"

#define NUM_THREADS 2
#define LOOP_ext 200
#define TAM_ARRAY 204800/LOOP_ext

int variable_compartida[TAM_ARRAY];

void work(void *threadid)
{
    int tid= sesc_self();

    int i=0,k=0;

    int *p;

    if(tid==1) {

        for(i=0;i<LOOP_ext;i++) {

            //DISPARO 0

```

```
p = (int *) 405000000;
*p=25;

for(k=0;k<TAM_ARRAY;k++) {
    variable_compartida[k]=i;
}

//DISPARO 1
p = (int *) 405000001;
*p=25;
}

} else if(tid==2) {

    for(i=0;i<LOOP_ext;i++) {

        //DISPARO 2
        p = (int *) 405000002;
        *p=25;

        for(k=0;k<TAM_ARRAY;k++) {
            variable_compartida[k]=i;
        }

        //DISPARO 3
        p = (int *) 405000003;
        *p=25;
    }

}
sesc_exit(0);
}

int main()
{
    int t;

    sesc_init();

    for(t=0;t<NUM_THREADS;t++) {
        sesc_spawn(work, (void*)t, SESC_FLAG_NOMIGRATE|SESC_FLAG_MAP|t+1);
    }
    sesc_exit(0);
}
```

**ESCRITOR/ESCRITOR (Matriz de Incidencia y marcado inicial)**

```
M0 = [1, 1, 0, 0, 1, 0]
```

$$I = \begin{vmatrix} -1, 1, 0, 0 \\ -1, 0, 0, 1 \\ 1, -1, 0, 0 \\ 0, 1, -1, 0 \\ 0, 0, -1, 1 \\ 0, 0, 1, -1 \end{vmatrix}$$

**Archivo de configuración**

```
1
6,4
1,1,0,0,1,0
-1,1,0,0#-1,0,0,1#1,-1,0,0#0,1,-1,0#0,0,-1,1#0,0,1,-1
4
1,0,0,0#0,1,0,0#0,0,1,0#0,0,0,1
```

**12.2.2.1 Mediciones Obtenidas con el Simulador Modificado****ARRAY 1**

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr.o
# File : sesc_TestWrWr.o.Kz9DFy: Mon Oct 17 19:25:26 2011
      Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
    766.653 KIPS  1.2476 MHz  58.770 secs  73.320 msec (rabbit)  73319766.000
Proc Avg.Time BPType Total     RAS     BPred     BTB     BTAC
  1 47.334 static  57.14% ( 50.00% of 14.29%) 58.33% (100.00% of 35.71%) 0.00%
  2 47.001 static  57.14% ( 50.00% of 14.29%) 58.33% (100.00% of 35.71%) 0.00%
      nInst BJ Load Store INT FP :LD Forward ,Replay :Worst Unit (clk)
  1 22528076 12.73% 20.91% 15.45% 47.27% 0.00% : 21.74% ??? inst/repl : LDSTIssueX 0.14
  2 22528117 12.73% 20.91% 15.45% 47.27% 0.00% : 21.74% ??? inst/repl : LDSTIssueX 0.12
Proc IPC Active   Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
  1 0.31 98.32 72090746 31.2 0.0 0.0 56.8 0.0 0.0 0.0 0.0 11.9 0.0 0.0
  2 0.31 98.32 72090915 31.2 0.0 0.0 57.1 0.0 0.0 0.0 0.0 11.6 0.0 0.0
#####
CACHE DE PETRIS
readHit   readMiss   writeHit   writeMiss
  0       0     819200     409599
#####
```

**ARRAY 2**

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr.o
# File : sesc_TestWrWr.o.GrXZqa: Mon Oct 17 19:34:11 2011
      Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
    782.294 KIPS  1.3171 MHz  33.510 secs  44.136 msec (rabbit)  44136040.000
Proc Avg.Time BPType Total     RAS     BPred     BTB     BTAC
  1 42.858 static  56.25% ( 50.00% of 12.50%) 57.14% (100.00% of 43.75%) 0.00%
  2 42.858 static  56.25% ( 50.00% of 12.50%) 57.14% (100.00% of 43.75%) 0.00%
      nInst BJ Load Store INT FP :LD Forward ,Replay :Worst Unit (clk)
  1 13107316 12.50% 21.09% 14.84% 48.44% 0.00% : 18.52% ??? inst/repl : LDSTIssueX 0.10
  2 13107367 12.50% 21.09% 14.84% 48.44% 0.00% : 18.52% ??? inst/repl : LDSTIssueX 0.10
Proc IPC Active   Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
  1 0.30 98.61 43521365 30.1 0.0 0.0 58.4 0.0 0.0 0.0 0.0 11.5 0.0 0.0
  2 0.30 98.61 43521590 30.1 0.0 0.0 58.4 0.0 0.0 0.0 0.0 11.5 0.0 0.0
```

```
#####
CACHE DE PETRIS
readHit    readMiss    writeHit    writeMiss
0          0        409600      204799
#####

ARRAY 4
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr.o
# File : sesc_TestWrWr.o.K6IPI9 : Mon Oct 17 19:38:01 2011
Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
798.352 KIPS 1.4181 MHz 28.090 secs 39.835 msec (rabbit) 39834974.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
1 53.112 static 71.87% (50.00% of 6.25%) 73.33% (100.00% of 46.88%) 0.00%
2 52.779 static 71.87% (50.00% of 6.25%) 73.33% (100.00% of 46.88%) 0.00%
nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
1 11212805 14.61% 26.94% 13.24% 43.38% 0.00% : 18.64% ??? inst/repl : LDSTIssueX 0.11
2 11212916 14.61% 26.94% 13.24% 43.38% 0.00% : 18.64% ??? inst/repl : LDSTIssueX 0.11
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.28 99.23 39527356 28.4 0.0 0.0 63.5 0.0 0.0 0.0 0.0 0.0 8.1 0.0 0.0
2 0.28 99.23 39527723 28.4 0.0 0.0 63.5 0.0 0.0 0.0 0.0 0.0 8.1 0.0 0.0
#####
CACHE DE PETRIS
readHit    readMiss    writeHit    writeMiss
0          0        204800      102399
#####

ARRAY 8
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr.o
# File : sesc_TestWrWr.o.0vZ2Ix : Mon Oct 17 19:39:40 2011
Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
785.291 KIPS 1.4379 MHz 22.950 secs 33.000 msec (rabbit) 32999752.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
1 53.848 static 75.93% (50.00% of 3.70%) 76.92% (100.00% of 48.15%) 0.00%
2 53.847 static 75.93% (50.00% of 3.70%) 76.92% (100.00% of 48.15%) 0.00%
nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
1 90111104 15.34% 29.26% 12.50% 41.76% 0.00% : 17.48% ??? inst/repl : LDSTIssueX 0.10
2 90111335 15.34% 29.26% 12.50% 41.76% 0.00% : 17.48% ??? inst/repl : LDSTIssueX 0.10
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.27 99.53 32845474 27.4 0.0 0.0 65.5 0.0 0.0 0.0 0.0 0.0 7.1 0.0 0.0
2 0.27 99.53 32846101 27.4 0.0 0.0 65.5 0.0 0.0 0.0 0.0 0.0 7.1 0.0 0.0
#####
CACHE DE PETRIS
readHit    readMiss    writeHit    writeMiss
0          0        102400      51199
#####

ARRAY 16
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr.o
# File : sesc_TestWrWr.o.EQelUB : Mon Oct 17 19:40:59 2011
Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
786.127 KIPS 1.4646 MHz 20.190 secs 29.569 msec (rabbit) 29569354.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
1 54.240 static 78.57% (50.00% of 2.04%) 79.17% (100.00% of 48.98%) 0.00%
2 54.239 static 78.57% (50.00% of 2.04%) 79.17% (100.00% of 48.98%) 0.00%
nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
1 7935764 15.81% 30.81% 11.94% 40.81% 0.00% : 16.75% ??? inst/repl : LDSTIssueX 0.10
2 7936135 15.81% 30.81% 11.94% 40.81% 0.00% : 16.75% ??? inst/repl : LDSTIssueX 0.10
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.27 99.74 29491359 26.9 0.0 0.0 66.7 0.0 0.0 0.0 0.0 0.0 6.4 0.0 0.0
2 0.27 99.74 29492504 26.9 0.0 0.0 66.7 0.0 0.0 0.0 0.0 0.0 6.4 0.0 0.0
#####
CACHE DE PETRIS
readHit    readMiss    writeHit    writeMiss
```

```

0      0    51200    25599
#####
ARRAY 32
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr.o
# File : sesc_TestWrWr.o.bavA1o : Mon Oct 17 19:42:24 2011
   Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
   791.604 KIPS  1.4914 MHz  18.530 secs  27.637 msec (rabbit)  27636551.000
Proc Avg.Time BPType   Total   RAS   BPred   BTB   BTAC
  1 54.597 static  79.89% (50.01% of 1.09%) 80.22% (100.00% of 49.46%) 0.00%
  2 54.596 static  79.89% (50.00% of 1.09%) 80.22% (100.00% of 49.46%) 0.00%
     nInst BJ Load Store INT FP : LD Forward ,Replay : Worst Unit (clk)
  1 7333894 16.06% 31.67% 11.61% 40.31% 0.00% : 16.25% ??? inst/repl : LDSTIssueX 0.10
  2 7334535 16.06% 31.68% 11.61% 40.31% 0.00% : 16.25% ??? inst/repl : LDSTIssueX 0.10
Proc IPC Active   Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
  1 0.27 99.85 27595957 26.6 0.0 0.0 67.4 0.0 0.0 0.0 0.0 0.0 6.0 0.0 0.0
  2 0.27 99.86 27598101 26.6 0.0 0.0 67.4 0.0 0.0 0.0 0.0 0.0 6.0 0.0 0.0
#####
CACHE DE PETRIS
readHit   readMiss   writeHit   writeMiss
  0      0    25600    12799
#####
ARRAY 64
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr.o
# File : sesc_TestWrWr.o.4057m7 : Mon Oct 17 19:44:19 2011
   Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
   785.827 KIPS  1.4896 MHz  17.900 secs  26.664 msec (rabbit)  26663789.000
Proc Avg.Time BPType   Total   RAS   BPred   BTB   BTAC
  1 54.785 static  80.61% (50.02% of 0.56%) 80.79% (100.00% of 49.72%) 0.00%
  2 54.770 static  80.62% (50.00% of 0.56%) 80.79% (100.00% of 49.72%) 0.00%
     nInst BJ Load Store INT FP : LD Forward ,Replay : Worst Unit (clk)
  1 7032564 16.20% 32.16% 11.42% 40.04% 0.00% : 15.98% ??? inst/repl : LDSTIssueX 0.09
  2 7033745 16.20% 32.17% 11.42% 40.04% 0.00% : 15.98% ??? inst/repl : LDSTIssueX 0.09
Proc IPC Active   Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
  1 0.26 99.91 26640396 26.4 0.0 0.0 67.8 0.0 0.0 0.0 0.0 0.0 5.8 0.0 0.0
  2 0.26 99.93 26644538 26.4 0.0 0.0 67.8 0.0 0.0 0.0 0.0 0.0 5.8 0.0 0.0
#####
CACHE DE PETRIS
readHit   readMiss   writeHit   writeMiss
  0      0    12800     6399
#####
ARRAY 128
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr.o
# File : sesc_TestWrWr.o.4ZSVf0 : Mon Oct 17 19:45:41 2011
   Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
   787.917 KIPS  1.4984 MHz  17.510 secs  26.237 msec (rabbit)  26236587.000
Proc Avg.Time BPType   Total   RAS   BPred   BTB   BTAC
  1 54.890 static  81.05% (50.03% of 0.29%) 81.13% (100.00% of 49.86%) 0.00%
  2 54.885 static  81.05% (50.00% of 0.28%) 81.14% (100.00% of 49.86%) 0.00%
     nInst BJ Load Store INT FP : LD Forward ,Replay : Worst Unit (clk)
  1 6897074 16.28% 32.44% 11.32% 39.87% 0.00% : 15.87% ??? inst/repl : LDSTIssueX 0.09
  2 6899345 16.28% 32.44% 11.32% 39.87% 0.00% : 15.87% ??? inst/repl : LDSTIssueX 0.09
Proc IPC Active   Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
  1 0.26 99.93 26218762 26.3 0.0 0.0 68.0 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.0
  2 0.26 99.96 26226936 26.3 0.0 0.0 68.0 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.0
#####
CACHE DE PETRIS
readHit   readMiss   writeHit   writeMiss
  0      0    6400     3199
#####

```

## ARRAY 256

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr.o
# File : sesc_TestWrWr.o.b10TPP : Mon Oct 17 20:07:10 2011
      Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
    788.674 KIPS  1.5024 MHz  17.320 secs  26.022 msec (rabbit)  26022189.000
Proc Avg.Time BPType   Total     RAS     BPred     BTB     BTAC
  1 54.941 static  81.26% (50.06% of 0.14%) 81.31% (100.00% of 49.93%) 0.00%
  2 54.937 static  81.28% (50.00% of 0.14%) 81.32% (100.00% of 49.93%) 0.00%
    nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
  1 6827694 16.32% 32.58% 11.27% 39.78% 0.00% : 15.80% ??? inst/repl : LDSTIssueX 0.09
  2 6832145 16.32% 32.59% 11.26% 39.78% 0.00% : 15.81% ??? inst/repl : LDSTIssueX 0.09
Proc IPC Active   Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
  1 0.26 99.92 26001099 26.3 0.0 0.0 68.1 0.0 0.0 0.0 0.0 5.6 0.0 0.0
  2 0.26 99.98 26017339 26.3 0.0 0.0 68.1 0.0 0.0 0.0 0.0 5.6 0.0 0.0
#####
CACHE DE PETRIS
readHit   readMiss   writeHit   writeMiss
  0        0       3200      1599
#####
```

## ARRAY 512

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr.o
# File : sesc_TestWrWr.o.mWg75m : Mon Oct 17 20:08:34 2011
      Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
    780.039 KIPS  1.4877 MHz  17.420 secs  25.916 msec (rabbit)  25915790.000
Proc Avg.Time BPType   Total     RAS     BPred     BTB     BTAC
  1 54.972 static  81.36% (50.12% of 0.07%) 81.38% (100.00% of 49.96%) 0.00%
  2 54.965 static  81.39% (50.00% of 0.07%) 81.41% (100.00% of 49.96%) 0.00%
    nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
  1 6789734 16.34% 32.65% 11.24% 39.75% 0.00% : 15.77% ??? inst/repl : LDSTIssueX 0.09
  2 6798545 16.35% 32.66% 11.24% 39.73% 0.00% : 15.78% ??? inst/repl : LDSTIssueX 0.09
Proc IPC Active   Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
  1 0.26 99.87 25880967 26.2 0.0 0.0 68.2 0.0 0.0 0.0 0.0 5.6 0.0 0.0
  2 0.26 99.99 25913340 26.2 0.0 0.0 68.2 0.0 0.0 0.0 0.0 5.6 0.0 0.0
#####
CACHE DE PETRIS
readHit   readMiss   writeHit   writeMiss
  0        0       1600      799
#####
```

## ARRAY 1024

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr.o
# File : sesc_TestWrWr.o.9YwTii : Mon Oct 17 20:09:55 2011
      Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
    790.243 KIPS  1.5088 MHz  17.140 secs  25.861 msec (rabbit)  25860994.000
Proc Avg.Time BPType   Total     RAS     BPred     BTB     BTAC
  1 54.983 static  81.35% (50.25% of 0.04%) 81.36% (100.00% of 49.96%) 0.00%
  2 54.981 static  81.41% (50.00% of 0.04%) 81.42% (100.00% of 49.96%) 0.00%
    nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
  1 6763617 16.35% 32.67% 11.23% 39.74% 0.00% : 15.73% ??? inst/repl : LDSTIssueX 0.09
  2 6781145 16.36% 32.69% 11.23% 39.72% 0.00% : 15.76% ??? inst/repl : LDSTIssueX 0.09
Proc IPC Active   Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
  1 0.26 99.75 25795111 26.2 0.0 0.0 68.2 0.0 0.0 0.0 0.0 5.6 0.0 0.0
  2 0.26 100.00 25859744 26.2 0.0 0.0 68.2 0.0 0.0 0.0 0.0 5.6 0.0 0.0
#####
CACHE DE PETRIS
readHit   readMiss   writeHit   writeMiss
  0        0       800       399
#####
```

### 12.2.2.2 Mediciones Obtenidas con el Simulador Original

ARRAY 1

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr-org.o
# File : sesc_TestWrWr-org.o.Ip2m7u : Mon Oct 17 19:50:10 2011
      Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
      808.306 KIPS  1.3560 MHz  69.930 secs  94.823 msec (rabbit)  94823463.000
Proc Avg.Time BPType  Total    RAS    BPred    BTB    BTAC
  1 54.000 static  72.73% (50.00% of 9.09%) 75.00% (100.00% of 40.91%) 0.00%
  2 54.000 static  72.73% (50.00% of 9.09%) 75.00% (100.00% of 40.91%) 0.00%
      nInst  BJ  Load  Store  INT  FP : LD Forward , Replay : Worst Unit (clk)
  1 28262376 15.94% 26.81% 12.32% 42.03% 0.00% : 18.92% ??? inst/repl : LDSTIssueX 0.10
  2 28262457 15.94% 26.81% 12.32% 42.03% 0.00% : 18.92% ??? inst/repl : LDSTIssueX 0.10
Proc IPC Active  Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
  1 0.30 100.00 94823188 29.8 0.0 0.0 61.1 0.0 0.0 0.0 0.0 9.0 0.0 0.0
  2 0.30 100.00 94823411 29.8 0.0 0.0 61.1 0.0 0.0 0.0 0.0 9.0 0.0 0.0
#####
CACHE DE PETRIS
readHit  readMiss  writeHit  writeMiss
  0        0        0        0
#####
```

ARRAY 2

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr-org.o
# File : sesc_TestWrWr-org.o.UhXdYL : Mon Oct 17 19:51:35 2011
      Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
      813.986 KIPS  1.3984 MHz  39.250 secs  54.888 msec (rabbit)  54887740.000
Proc Avg.Time BPType  Total    RAS    BPred    BTB    BTAC
  1 48.572 static  70.83% (50.00% of 8.33%) 72.73% (100.00% of 45.83%) 0.00%
  2 48.572 static  70.83% (50.00% of 8.33%) 72.73% (100.00% of 45.83%) 0.00%
      nInst  BJ  Load  Store  INT  FP : LD Forward , Replay : Worst Unit (clk)
  1 15974426 15.38% 26.28% 12.18% 43.59% 0.00% : 17.07% ??? inst/repl : LDSTIssueX 0.09
  2 15974517 15.38% 26.28% 12.18% 43.59% 0.00% : 17.07% ??? inst/repl : LDSTIssueX 0.09
Proc IPC Active  Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
  1 0.29 100.00 54887440 29.1 0.0 0.0 61.8 0.0 0.0 0.0 0.0 9.1 0.0 0.0
  2 0.29 100.00 54887688 29.1 0.0 0.0 61.8 0.0 0.0 0.0 0.0 9.1 0.0 0.0
#####
CACHE DE PETRIS
readHit  readMiss  writeHit  writeMiss
  0        0        0        0
#####
```

ARRAY 4

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr-org.o
# File : sesc_TestWrWr-org.o.55J8pF : Mon Oct 17 19:53:56 2011
      Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
      798.163 KIPS  1.4180 MHz  27.840 secs  39.476 msec (rabbit)  39476464.000
Proc Avg.Time BPType  Total    RAS    BPred    BTB    BTAC
  1 49.779 static  73.53% (50.00% of 5.88%) 75.00% (100.00% of 47.06%) 0.00%
  2 49.779 static  73.53% (50.00% of 5.88%) 75.00% (100.00% of 47.06%) 0.00%
      nInst  BJ  Load  Store  INT  FP : LD Forward , Replay : Worst Unit (clk)
  1 11110365 15.67% 28.11% 11.98% 42.40% 0.00% : 16.39% ??? inst/repl : LDSTIssueX 0.09
  2 11110506 15.67% 28.11% 11.98% 42.40% 0.00% : 16.39% ??? inst/repl : LDSTIssueX 0.09
Proc IPC Active  Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
  1 0.28 100.00 39476026 28.1 0.0 0.0 63.7 0.0 0.0 0.0 0.0 8.2 0.0 0.0
  2 0.28 100.00 39476412 28.1 0.0 0.0 63.7 0.0 0.0 0.0 0.0 8.2 0.0 0.0
#####
CACHE DE PETRIS
readHit  readMiss  writeHit  writeMiss
  0        0        0        0
#####
```

#### ARRAY 8

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr-org.o
# File : sesc_TestWrWr-org.o.BuGnZY : Mon Oct 17 19:55:23 2011
Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
782.192 KIPS 1.4326 MHz 22.910 secs 32.821 msec (rabbit) 32820546.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
1 51.771 static 76.78% (50.00% of 3.57%) 77.78% (100.00% of 48.21%) 0.00%
2 51.771 static 76.79% (50.00% of 3.57%) 77.78% (100.00% of 48.21%) 0.00%
  nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
1 8959924 16.00% 30.00% 11.71% 41.14% 0.00% : 16.19% ??? inst/repl : LDSTIssueX 0.09
2 8960105 16.00% 30.00% 11.71% 41.14% 0.00% : 16.19% ??? inst/repl : LDSTIssueX 0.09
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.27 100.00 32819870 27.3 0.0 0.0 65.6 0.0 0.0 0.0 0.0 0.0 7.1 0.0 0.0
2 0.27 100.00 32820494 27.3 0.0 0.0 65.6 0.0 0.0 0.0 0.0 0.0 7.1 0.0 0.0
#####
CACHE DE PETRIS
readHit readMiss writeHit writeMiss
0 0 0 0
#####
```

#### ARRAY 16

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr-org.o
# File : sesc_TestWrWr-org.o.xVJcCm : Mon Oct 17 19:56:23 2011
Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
797.011 KIPS 1.4858 MHz 19.850 secs 29.493 msec (rabbit) 29492509.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
1 53.002 static 79.00% (50.00% of 2.00%) 79.59% (100.00% of 49.00%) 0.00%
2 53.002 static 79.00% (50.00% of 2.00%) 79.59% (100.00% of 49.00%) 0.00%
  nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
1 7910174 16.18% 31.23% 11.49% 40.45% 0.00% : 16.06% ??? inst/repl : LDSTIssueX 0.09
2 7910495 16.18% 31.23% 11.49% 40.45% 0.00% : 16.06% ??? inst/repl : LDSTIssueX 0.09
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.27 100.00 29491315 26.8 0.0 0.0 66.8 0.0 0.0 0.0 0.0 0.0 6.4 0.0 0.0
2 0.27 100.00 29492458 26.8 0.0 0.0 66.8 0.0 0.0 0.0 0.0 0.0 6.4 0.0 0.0
#####
CACHE DE PETRIS
readHit readMiss writeHit writeMiss
0 0 0 0
#####
```

#### ARRAY 32

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr-org.o
# File : sesc_TestWrWr-org.o.ukLZDO : Mon Oct 17 19:58:06 2011
Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
791.504 KIPS 1.4914 MHz 18.500 secs 27.592 msec (rabbit) 27591746.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
1 53.867 static 80.11% (50.01% of 1.08%) 80.43% (100.00% of 49.46%) 0.00%
2 53.867 static 80.11% (50.00% of 1.08%) 80.43% (100.00% of 49.46%) 0.00%
  nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
1 7321114 16.26% 31.91% 11.36% 40.12% 0.00% : 15.89% ??? inst/repl : LDSTIssueX 0.09
2 7321705 16.26% 31.91% 11.36% 40.12% 0.00% : 15.89% ??? inst/repl : LDSTIssueX 0.09
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.27 99.99 27589553 26.5 0.0 0.0 67.5 0.0 0.0 0.0 0.0 0.0 6.0 0.0 0.0
2 0.27 100.00 27591695 26.5 0.0 0.0 67.5 0.0 0.0 0.0 0.0 0.0 6.0 0.0 0.0
#####
CACHE DE PETRIS
readHit readMiss writeHit writeMiss
0 0 0 0
#####
```

## ARRAY 64

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr-org.o
# File : sesc_TestWrWr-org.o.YWikqe : Mon Oct 17 19:59:24 2011
Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
788.635 KIPS 1.4950 MHz 17.820 secs 26.641 msec (rabbit) 26641346.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
1 54.394 static 80.72% (50.02% of 0.56%) 80.89% (100.00% of 49.72%) 0.00%
2 54.393 static 80.73% (50.00% of 0.56%) 80.90% (100.00% of 49.72%) 0.00%
nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
1 7026174 16.30% 32.28% 11.29% 39.94% 0.00% : 15.80% ??? inst/repl : LDSTIssueX 0.09
2 7027305 16.30% 32.29% 11.29% 39.94% 0.00% : 15.80% ??? inst/repl : LDSTIssueX 0.09
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.26 99.98 26637155 26.4 0.0 0.0 67.8 0.0 0.0 0.0 0.0 0.0 5.8 0.0 0.0
2 0.26 100.00 26641295 26.4 0.0 0.0 67.8 0.0 0.0 0.0 0.0 0.0 5.8 0.0 0.0
#####
CACHE DE PETRIS
readHit readMiss writeHit writeMiss
0 0 0 0
#####
```

## ARRAY 128

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr-org.o
# File : sesc_TestWrWr-org.o.sECRLR : Mon Oct 17 20:01:11 2011
Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
792.073 KIPS 1.5063 MHz 17.410 secs 26.225 msec (rabbit) 26225346.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
1 54.687 static 81.10% (50.03% of 0.28%) 81.19% (100.00% of 49.86%) 0.00%
2 54.686 static 81.11% (50.00% of 0.28%) 81.20% (100.00% of 49.86%) 0.00%
nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
1 6893884 16.33% 32.50% 11.25% 39.82% 0.00% : 15.77% ??? inst/repl : LDSTIssueX 0.09
2 6896105 16.33% 32.51% 11.25% 39.81% 0.00% : 15.77% ??? inst/repl : LDSTIssueX 0.09
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.26 99.97 26217122 26.3 0.0 0.0 68.0 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.0
2 0.26 100.00 26225294 26.3 0.0 0.0 68.0 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.0
#####
CACHE DE PETRIS
readHit readMiss writeHit writeMiss
0 0 0 0
#####
```

## ARRAY 256

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr-org.o
# File : sesc_TestWrWr-org.o.raVu9Q : Mon Oct 17 20:01:49 2011
Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
796.768 KIPS 1.5179 MHz 17.140 secs 26.017 msec (rabbit) 26017346.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
1 54.842 static 81.29% (50.06% of 0.14%) 81.33% (100.00% of 49.93%) 0.00%
2 54.841 static 81.30% (50.00% of 0.14%) 81.35% (100.00% of 49.93%) 0.00%
nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
1 6826104 16.35% 32.61% 11.23% 39.76% 0.00% : 15.76% ??? inst/repl : LDSTIssueX 0.09
2 6830505 16.35% 32.62% 11.23% 39.75% 0.00% : 15.76% ??? inst/repl : LDSTIssueX 0.09
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.26 99.94 26001056 26.3 0.0 0.0 68.1 0.0 0.0 0.0 0.0 0.0 5.6 0.0 0.0
2 0.26 100.00 26017294 26.3 0.0 0.0 68.1 0.0 0.0 0.0 0.0 0.0 5.6 0.0 0.0
#####
CACHE DE PETRIS
readHit readMiss writeHit writeMiss
0 0 0 0
#####
```

#### ARRAY 512

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr-org.o
# File : sesc_TestWrWr-org.o.v8cZOH : Mon Oct 17 20:03:31 2011
Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
785.810 KIPS 1.4987 MHz 17.290 secs 25.913 msec (rabbit) 25913346.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
1 54.922 static 81.37% (50.12% of 0.07%) 81.40% (100.00% of 49.96%) 0.00%
2 54.921 static 81.40% (50.00% of 0.07%) 81.42% (100.00% of 49.96%) 0.00%
  nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
1 6788944 16.35% 32.67% 11.22% 39.73% 0.00% : 15.74% ??? inst/repl : LDSTIssueX 0.09
2 6797705 16.36% 32.68% 11.22% 39.72% 0.00% : 15.76% ??? inst/repl : LDSTIssueX 0.09
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.26 99.87 25880924 26.2 0.0 0.0 68.2 0.0 0.0 0.0 0.0 0.0 5.6 0.0 0.0
2 0.26 100.00 25913294 26.2 0.0 0.0 68.2 0.0 0.0 0.0 0.0 0.0 5.6 0.0 0.0
#####
CACHE DE PETRIS
readHit readMiss writeHit writeMiss
0 0 0 0
#####
```

#### ARRAY 1024

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestWrWr-org.o
# File : sesc_TestWrWr-org.o.pqLo3n : Mon Oct 17 20:04:57 2011
Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
807.147 KIPS 1.5411 MHz 16.780 secs 25.860 msec (rabbit) 25859947.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
1 54.958 static 81.36% (50.25% of 0.04%) 81.37% (100.00% of 49.96%) 0.00%
2 54.958 static 81.42% (50.00% of 0.04%) 81.43% (100.00% of 49.96%) 0.00%
  nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
1 6763227 16.35% 32.68% 11.22% 39.74% 0.00% : 15.72% ??? inst/repl : LDSTIssueX 0.09
2 6780705 16.36% 32.70% 11.22% 39.71% 0.00% : 15.75% ??? inst/repl : LDSTIssueX 0.09
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.26 99.75 25795267 26.2 0.0 0.0 68.2 0.0 0.0 0.0 0.0 0.0 5.6 0.0 0.0
2 0.26 100.00 25859896 26.2 0.0 0.0 68.2 0.0 0.0 0.0 0.0 0.0 5.6 0.0 0.0
#####
CACHE DE PETRIS
readHit readMiss writeHit writeMiss
0 0 0 0
#####
```

### 12.2.3 PRODUCTOR/CONSUMIDOR

**PRODUCTOR/CONSUMIDOR** (para correr con simulador original)

```
#include "$HOME/esesc/src/libapp/sescapi.h"
#include "$HOME/esesc/src/libapp/sesc_locks.c"
#include "$HOME/esesc/src/libapp/sesc_events.c"

#define NUM_THREADS 2
//#define N 15           // Elementos en la cola de producción
#define LOOP_ext 20000

ssema_t semactrlC;    // Se inicializan a 0 todos
ssema_t semadistwar;
ssema_t semadistmenor;
ssema_t semasuspend;

int entrada = 0;
int salida = 0;
int buffer[N];
int Producciones = 0;
int Consumiciones = 0;

void work(void *threadid)
{
    int tid= sesc_self();
    int item,k;

    for(k=0;k<LOOP_ext;k++) {
        if((tid==1) || (tid==3) || (tid==5)) {

            item =1;
            sesc_psema(&semavacio);
            sesc_psema(&semamutex);

            buffer[entrada] = item;
            entrada = (entrada + 1) % N;
            //fprintf(stderr,"PRODUCIENDO %d\n", tid);
            Producciones=Producciones +1;
            sesc_vsema(&semamutex);
            sesc_vsema(&semalleno);
        }

        if((tid==2) || (tid==4) || (tid==6)) {

            sesc_psema(&semalleno);
        }
    }
}
```

```

        sesc_psema(&semamutex);
        item = buffer[salida];
        salida = (salida + 1) % N;
        Consumiciones = Consumiciones +1;
        //fprintf(stderr,"CONSUMIENDO      %d\n", tid);
        sesc_vsema(&semamutex);
        sesc_vsema(&semavacio);
    }
}

fprintf(stderr,"PRODUCCIONES =      %d\n", Producciones);
fprintf(stderr,"CONSUMICIONES =      %d\n", Consumiciones);
sesc_exit(0);
}

int main() {

    int t;
    sesc_init();

    sesc_sema_init(&semactrlC , (int32_t) 0);
    sesc_sema_init(&semadistwar , (int32_t) 0);
    sesc_sema_init(&semadistmenor , (int32_t) 0);
    sesc_sema_init(&semasuspend , (int32_t) 0);

    for(t=0;t<NUM_THREADS;t++) {
        //printf("Creating thread %d\n", t);
        sesc_spawn(work, (void *) t, SESC_FLAG_NOMIGRATE|SESC_FLAG_MAP|t+1);
    }

    sesc_exit(0);

}

```

### PRODUCTOR/CONSUMIDOR (para correr con simulador que implementa Petri)

```

#include <stdio.h>
#include <stdlib.h>
#include "/home/fede/esesc/src/libapp/sescapi.h"
#include "/home/fede/esesc/src/libapp/sesc_locks.c"
#include "/home/fede/esesc/src/libapp/sesc_events.c"

#define NUM_THREADS 2
#define N 15
#define LOOP_ext 200000

ssema_t semamutex; // Se inicializa a 1

```

```

ssema_t semalleno; // Se inicializa a 0
ssema_t semavacio; // Se inicializa a N

int entrada = 0;
int salida = 0;
int buffer[N];
int contador = 0;

void work(void *threadid)
{
    int tid= sesc_self();
    int item,k,l;

    if(tid==1) {

        for(k=0;k<LOOP_ext;k++) {

            item = 5;

            //DISPARO 0
            p = (int *) 405000000;
            *p=25;

            buffer[entrada] = 5;//item;
            entrada = (entrada + 1) % N;

            //DISPARO 1
            p = (int *) 405000001;
            *p=25;
        }
    }

    }else(tid==2) {

        for(l=0;l<LOOP_ext;l++) {

            //DISPARO 2
            p = (int *) 405000002;
            *p=25;

            item = buffer[salida];
            salida = (salida + 1) % N;

            //DISPARO 3
            p = (int *) 405000003;
            *p=25;

            item=0;
    }
}

```

```

        }
    }

    sesc_exit(0);
}

int main() {
    int t;
    sesc_init();

    sesc_sema_init(&semamutex , (int32_t) 1);
    sesc_sema_init(&semalleno , (int32_t) 0);
    sesc_sema_init(&semavacio , (int32_t) 25);

    for(t=0;t<NUM_THREADS;t++) {

        //printf("Creating thread %d\n", t);
        sesc_spawn(work, (void *) t, SESC_FLAG_NOMIGRATE|SESC_FLAG_MAP|t+1);

    }
    sesc_exit(0);
}

```

**PRODUCTOR/CONSUMIDOR (Matriz de Incidencia y marcado inicial)**

$M_0 = [1, 0, 1, 10, 0, 1, 0]$

$$I = \begin{vmatrix} -1, 1, 0, 0 \\ 1, -1, 0, 0 \\ -1, 1, -1, 1 \\ -1, 0, 0, 1 \\ 0, 1, -1, 0 \\ 0, 0, -1, 1 \\ 0, 0, 1, -1 \end{vmatrix}$$

**Archivo de configuración**

```

1
7,4
1,0,1,10,0,1,0
-1,1,0,0#1,-1,0,0#-1,1,-1,1#-1,0,0,1#0,1,-1,0#0,0,-1,1#0,0,1,-1
4
1,0,0,0#0,1,0,0#0,0,1,0#0,0,0,1

```

**12.2.3.1 Mediciones Obtenidas con el Simulador Modificado**

```

LOOP 10000
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestPrCo.o
# File : sesc_TestPrCo.o.Lj0VA1 : Tue Oct 18 01:13:36 2011
      Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles

```

767.671 KIPS	0.4128 MHz	11.630 secs	4.801 msec (rabbit)	4801039.000	
Proc Avg.Time BPType	Total	RAS	BPred	BTB	BTAC
1 67.169 static	73.65% ( 97.89% of 32.40%)	62.03% ( 99.98% of 41.22%)	0.00%		
2 69.934 static	74.98% ( 99.94% of 33.27%)	62.53% ( 99.98% of 41.66%)	0.00%		
3 67.173 static	73.66% ( 97.89% of 32.38%)	62.05% ( 99.98% of 41.22%)	0.00%		
4 69.957 static	74.98% ( 99.95% of 33.27%)	62.53% ( 99.98% of 41.66%)	0.00%		
5 67.170 static	73.66% ( 97.89% of 32.38%)	62.05% ( 99.98% of 41.22%)	0.00%		
6 70.004 static	74.98% ( 99.95% of 33.28%)	62.52% ( 99.98% of 41.66%)	0.00%		
	nInst	BJ	Load Store	INT	FP : LD Forward , Replay : Worst Unit (clk)
1	1494850	8.26%	17.63%	14.16%	54.59% 0.00% : 15.52% ??? inst/repl : LDSTIssueX 0.06
2	1481104	8.12%	18.25%	14.19%	54.04% 0.00% : 14.83% ??? inst/repl : LDSTIssueX 0.06
3	1495094	8.26%	17.64%	14.16%	54.59% 0.00% : 15.52% ??? inst/repl : LDSTIssueX 0.06
4	1480996	8.12%	18.25%	14.19%	54.04% 0.00% : 14.83% ??? inst/repl : LDSTIssueX 0.06
5	1495098	8.26%	17.64%	14.16%	54.59% 0.00% : 15.52% ??? inst/repl : LDSTIssueX 0.06
6	1480872	8.12%	18.25%	14.19%	54.04% 0.00% : 14.83% ??? inst/repl : LDSTIssueX 0.06
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other					
1 0.35 89.60	4301844	34.7	0.0	0.0	60.0 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
2 0.34 89.63	4303335	34.4	0.0	0.0	60.7 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0
3 0.35 89.62	4302689	34.7	0.0	0.0	60.0 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
4 0.34 89.62	4302903	34.4	0.0	0.0	60.7 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0
5 0.35 89.62	4302690	34.7	0.0	0.0	60.0 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
6 0.34 89.62	4302474	34.4	0.0	0.0	60.7 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0
#####					
CACHE DE PETRIS					
readHit	readMiss	writeHit	writeMiss		
0	0	120000	377651		
#####					

LOOP 15000					
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestPrCo.o					
# File : sesc_TestPrCo.o.TC9OGL : Tue Oct 18 01:16:10 2011					
Exe Speed	Exe MHz	Exe Time	Sim Time (1000MHz)	Cycles	
758.599 KIPS	0.4079 MHz	17.650 secs	7.200 msec (rabbit)	7200211.000	
Proc Avg.Time BPType	Total	RAS	BPred	BTB	BTAC
1 67.198 static	73.65% ( 97.90% of 32.41%)	62.02% ( 99.99% of 41.22%)	0.00%		
2 69.987 static	74.98% ( 99.96% of 33.29%)	62.52% ( 99.99% of 41.66%)	0.00%		
3 67.198 static	73.65% ( 97.90% of 32.40%)	62.03% ( 99.99% of 41.22%)	0.00%		
4 70.003 static	74.99% ( 99.97% of 33.29%)	62.52% ( 99.99% of 41.66%)	0.00%		
5 67.196 static	73.65% ( 97.90% of 32.40%)	62.03% ( 99.99% of 41.22%)	0.00%		
6 70.034 static	74.99% ( 99.97% of 33.30%)	62.52% ( 99.99% of 41.66%)	0.00%		
	nInst	BJ	Load Store	INT	FP : LD Forward , Replay : Worst Unit (clk)
1	2241922	8.26%	17.63%	14.17%	54.59% 0.00% : 15.51% ??? inst/repl : LDSTIssueX 0.06
2	2221104	8.11%	18.25%	14.19%	54.04% 0.00% : 14.82% ??? inst/repl : LDSTIssueX 0.06
3	2242183	8.26%	17.63%	14.16%	54.59% 0.00% : 15.51% ??? inst/repl : LDSTIssueX 0.06
4	2220996	8.11%	18.25%	14.19%	54.05% 0.00% : 14.82% ??? inst/repl : LDSTIssueX 0.06
5	2242187	8.26%	17.63%	14.16%	54.59% 0.00% : 15.51% ??? inst/repl : LDSTIssueX 0.06
6	2220872	8.11%	18.25%	14.19%	54.05% 0.00% : 14.82% ??? inst/repl : LDSTIssueX 0.06
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other					
1 0.35 89.59	6450964	34.8	0.0	0.0	60.0 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
2 0.34 89.62	6452505	34.4	0.0	0.0	60.7 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0
3 0.35 89.61	6451885	34.8	0.0	0.0	60.0 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
4 0.34 89.61	6452071	34.4	0.0	0.0	60.7 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0
5 0.35 89.61	6451886	34.8	0.0	0.0	60.0 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
6 0.34 89.60	6451642	34.4	0.0	0.0	60.7 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0
#####					
CACHE DE PETRIS					
readHit	readMiss	writeHit	writeMiss		
0	0	180000	567651		
#####					

LOOP 20000

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestPrCo.o
# File : sesc_TestPrCo.o.t94oH6 : Tue Oct 18 01:17:37 2011
      Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
      753.186 KIPS  0.4050 MHz  23.700 secs   9.599 msec (rabbit)  9599374.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
 1 67.209 static  73.65% (97.91% of 32.41%) 62.01% (99.99% of 41.22%) 0.00%
 2 70.013 static  74.99% (99.97% of 33.30%) 62.52% (99.99% of 41.66%) 0.00%
 3 67.211 static  73.65% (97.91% of 32.41%) 62.03% (99.99% of 41.22%) 0.00%
 4 70.025 static  74.99% (99.97% of 33.30%) 62.52% (99.99% of 41.66%) 0.00%
 5 67.210 static  73.65% (97.91% of 32.41%) 62.02% (99.99% of 41.22%) 0.00%
 6 70.048 static  74.99% (99.98% of 33.31%) 62.51% (99.99% of 41.66%) 0.00%
      nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
 1 2989011 8.26% 17.63% 14.17% 54.60% 0.00% : 15.51% ??? inst/repl : LDSTIssueX 0.06
 2 2961104 8.11% 18.25% 14.19% 54.05% 0.00% : 14.82% ??? inst/repl : LDSTIssueX 0.06
 3 2989255 8.26% 17.63% 14.17% 54.59% 0.00% : 15.51% ??? inst/repl : LDSTIssueX 0.06
 4 2960996 8.11% 18.25% 14.19% 54.05% 0.00% : 14.82% ??? inst/repl : LDSTIssueX 0.06
 5 2989259 8.26% 17.63% 14.17% 54.59% 0.00% : 15.51% ??? inst/repl : LDSTIssueX 0.06
 6 2960872 8.11% 18.25% 14.19% 54.05% 0.00% : 14.82% ??? inst/repl : LDSTIssueX 0.06
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
 1 0.35 89.59 8600148 34.8 0.0 0.0 60.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
 2 0.34 89.61 8601667 34.4 0.0 0.0 60.7 0.0 0.0 0.0 0.0 4.8 0.0 0.0
 3 0.35 89.60 8600997 34.8 0.0 0.0 60.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
 4 0.34 89.60 8601234 34.4 0.0 0.0 60.7 0.0 0.0 0.0 0.0 4.8 0.0 0.0
 5 0.35 89.60 8600998 34.8 0.0 0.0 60.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
 6 0.34 89.60 8600804 34.4 0.0 0.0 60.7 0.0 0.0 0.0 0.0 4.8 0.0 0.0
#####
CACHE DE PETRIS
readHit      readMiss      writeHit      writeMiss
 0          0        240000      757651
#####
```

LOOP 30000

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestPrCo.o
# File : sesc_TestPrCo.o.y7mh65 : Tue Oct 18 01:19:13 2011
      Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
      765.162 KIPS  0.4115 MHz  34.990 secs  14.398 msec (rabbit)  14397709.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
 1 67.223 static  73.65% (97.91% of 32.42%) 62.01% (99.99% of 41.22%) 0.00%
 2 70.040 static  74.99% (99.98% of 33.31%) 62.51% (99.99% of 41.66%) 0.00%
 3 67.223 static  73.65% (97.91% of 32.42%) 62.02% (99.99% of 41.22%) 0.00%
 4 70.048 static  74.99% (99.98% of 33.31%) 62.51% (99.99% of 41.67%) 0.00%
 5 67.222 static  73.65% (97.91% of 32.42%) 62.02% (99.99% of 41.22%) 0.00%
 6 70.063 static  74.99% (99.98% of 33.32%) 62.51% (99.99% of 41.67%) 0.00%
      nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
 1 4483172 8.26% 17.63% 14.17% 54.60% 0.00% : 15.51% ??? inst/repl : LDSTIssueX 0.06
 2 4441104 8.11% 18.25% 14.19% 54.05% 0.00% : 14.82% ??? inst/repl : LDSTIssueX 0.06
 3 4483433 8.26% 17.63% 14.17% 54.60% 0.00% : 15.51% ??? inst/repl : LDSTIssueX 0.06
 4 4440996 8.11% 18.25% 14.19% 54.05% 0.00% : 14.82% ??? inst/repl : LDSTIssueX 0.06
 5 4483437 8.26% 17.63% 14.17% 54.60% 0.00% : 15.51% ??? inst/repl : LDSTIssueX 0.06
 6 4440872 8.11% 18.25% 14.19% 54.05% 0.00% : 14.82% ??? inst/repl : LDSTIssueX 0.06
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
 1 0.35 89.59 12898464 34.8 0.0 0.0 60.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
 2 0.34 89.60 12900003 34.4 0.0 0.0 60.7 0.0 0.0 0.0 0.0 4.8 0.0 0.0
 3 0.35 89.59 12899387 34.8 0.0 0.0 60.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
 4 0.34 89.59 12899574 34.4 0.0 0.0 60.7 0.0 0.0 0.0 0.0 4.8 0.0 0.0
 5 0.35 89.59 12899387 34.8 0.0 0.0 60.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
 6 0.34 89.59 12899142 34.4 0.0 0.0 60.7 0.0 0.0 0.0 0.0 4.8 0.0 0.0
#####
CACHE DE PETRIS
readHit      readMiss      writeHit      writeMiss
 0          0        360000      1137651
#####
```

LOOP 40000

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestPrCo.o
# File : sesc_TestPrCo.o.o.lTZyp : Tue Oct 18 01:21:26 2011
```

Proc	Avg.Time	BPTType	Exe Speed	Exe MHz	Exe Time	Sim Time (1000MHz)	Cycles				
			Total	RAS	BPred	BTB	BTAC				
1	65.556	static	779.600 KIPS	0.4173 MHz	46.090 secs	19.235 msec (rabbit)	19235293.000				
2	68.896	static									
3	65.556	static									
4	68.895	static									
5	65.553	static									
6	68.894	static									
	nInst	BJ		Load	Store	INT	FP : LD Forward , Replay : Worst Unit (clk)				
1	6016942	8.20%	17.51%	14.07%	54.90%	0.00%	: 15.51% ???? inst/repl : LDSTIssueX 0.06				
2	5960348	8.05%	18.12%	14.09%	54.36%	0.00%	: 14.82% ???? inst/repl : LDSTIssueX 0.06				
3	6016946	8.20%	17.51%	14.07%	54.90%	0.00%	: 15.51% ???? inst/repl : LDSTIssueX 0.06				
4	5960358	8.05%	18.12%	14.09%	54.36%	0.00%	: 14.82% ???? inst/repl : LDSTIssueX 0.06				
5	6016840	8.20%	17.51%	14.07%	54.90%	0.00%	: 15.51% ???? inst/repl : LDSTIssueX 0.06				
6	5960342	8.05%	18.12%	14.09%	54.36%	0.00%	: 14.82% ???? inst/repl : LDSTIssueX 0.06				
Proc	IPC	Active	Cycles	Busy	LDQ	STQ	IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other				
1	0.35	89.60	17235251	34.9	0.0	0.0	59.8 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0				
2	0.35	89.60	17235502	34.6	0.0	0.0	60.6 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0				
3	0.35	89.60	17235251	34.9	0.0	0.0	59.8 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0				
4	0.35	89.60	17235502	34.6	0.0	0.0	60.6 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0				
5	0.35	89.60	17234911	34.9	0.0	0.0	59.8 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0				
6	0.35	89.60	17235500	34.6	0.0	0.0	60.6 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0				
# #####											
CACHE DE PETRIS											
readHit		readMiss		writeHit		writeMiss					
0	0	480000		1519739							
# #####											

LOOP 50000

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestPrCo.o
# File : sesc_TestPrCo.o.3ocTEM : Tue Oct 18 01:23:45 2011
```

Proc	Avg.Time	BPTType	Exe Speed	Exe MHz	Exe Time	Sim Time (1000MHz)	Cycles				
			Total	RAS	BPred	BTB	BTAC				
1	65.556	static	768.029 KIPS	0.4111 MHz	58.480 secs	24.044 msec (rabbit)	24043630.000				
2	68.896	static									
3	65.555	static									
4	68.896	static									
5	65.553	static									
6	68.895	static									
	nInst	BJ		Load	Store	INT	FP : LD Forward , Replay : Worst Unit (clk)				
1	7521120	8.20%	17.51%	14.07%	54.90%	0.00%	: 15.51% ???? inst/repl : LDSTIssueX 0.06				
2	7450348	8.05%	18.12%	14.09%	54.36%	0.00%	: 14.82% ???? inst/repl : LDSTIssueX 0.06				
3	7521124	8.20%	17.51%	14.07%	54.90%	0.00%	: 15.51% ???? inst/repl : LDSTIssueX 0.06				
4	7450358	8.05%	18.12%	14.09%	54.36%	0.00%	: 14.82% ???? inst/repl : LDSTIssueX 0.06				
5	7521018	8.20%	17.51%	14.07%	54.90%	0.00%	: 15.51% ???? inst/repl : LDSTIssueX 0.06				
6	7450342	8.05%	18.12%	14.09%	54.36%	0.00%	: 14.82% ???? inst/repl : LDSTIssueX 0.06				
Proc	IPC	Active	Cycles	Busy	LDQ	STQ	IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other				
1	0.35	89.60	21543636	34.9	0.0	0.0	59.8 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0				
2	0.35	89.60	21543838	34.6	0.0	0.0	60.6 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0				
3	0.35	89.60	21543637	34.9	0.0	0.0	59.8 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0				
4	0.35	89.60	21543839	34.6	0.0	0.0	60.6 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0				
5	0.35	89.60	21543275	34.9	0.0	0.0	59.8 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0				
6	0.35	89.60	21543838	34.6	0.0	0.0	60.6 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0				
# #####											
CACHE DE PETRIS											
readHit		readMiss		writeHit		writeMiss					
0	0	600000		1899739							
# #####											

```

LOOP 100000
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestPrCo.o
# File : sesc_TestPrCo.o.0QrTuO : Tue Oct 18 01:26:27 2011
      Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
      779.196 KIPS  0.4135 MHz  116.050 secs  47.991 msec (rabbit)  47990559.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
 1 65.699 static   73.65% (97.92% of 32.43%) 62.00% (100.00% of 41.22%) 0.00%
 2 68.892 static   75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
 3 65.699 static   73.65% (97.92% of 32.43%) 62.00% (100.00% of 41.22%) 0.00%
 4 68.892 static   75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
 5 65.699 static   73.65% (97.92% of 32.43%) 62.00% (100.00% of 41.22%) 0.00%
 6 68.892 static   75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
      nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
 1 15141682 8.15% 17.39% 13.98% 55.20% 0.00% : 15.51% ??? inst/repl : LDSTIssueX 0.06
 2 15000198 8.00% 18.00% 14.00% 54.67% 0.00% : 14.82% ??? inst/repl : LDSTIssueX 0.06
 3 15141686 8.15% 17.39% 13.98% 55.20% 0.00% : 15.51% ??? inst/repl : LDSTIssueX 0.06
 4 15000208 8.00% 18.00% 14.00% 54.67% 0.00% : 14.82% ??? inst/repl : LDSTIssueX 0.06
 5 15141690 8.15% 17.39% 13.98% 55.20% 0.00% : 15.51% ??? inst/repl : LDSTIssueX 0.06
 6 15000198 8.00% 18.00% 14.00% 54.67% 0.00% : 14.82% ??? inst/repl : LDSTIssueX 0.06
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
 1 0.35 90.00 43189889 35.1 0.0 0.0 59.7 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
 2 0.35 90.00 43190538 34.7 0.0 0.0 60.5 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0
 3 0.35 90.00 43189889 35.1 0.0 0.0 59.7 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
 4 0.35 90.00 43190538 34.7 0.0 0.0 60.5 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0
 5 0.35 90.00 43189886 35.1 0.0 0.0 59.7 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
 6 0.35 90.00 43190538 34.7 0.0 0.0 60.5 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0
#####
CACHE DE PETRIS
readHit      readMiss      writeHit      writeMiss
 0          0        1200000    3599968
#####

```

```

LOOP 200000
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestPrCo.o
# File : sesc_TestPrCo.o.rSpWc0 : Tue Oct 18 01:31:14 2011
      Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
      779.461 KIPS  0.4137 MHz  232.020 secs  95.979 msec (rabbit)  95979451.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
 1 65.699 static   73.65% (97.92% of 32.43%) 62.00% (100.00% of 41.22%) 0.00%
 2 68.892 static   75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
 3 65.699 static   73.65% (97.92% of 32.43%) 62.00% (100.00% of 41.22%) 0.00%
 4 68.892 static   75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
 5 65.699 static   73.65% (97.92% of 32.43%) 62.00% (100.00% of 41.22%) 0.00%
 6 68.892 static   75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
      nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
 1 30283343 8.15% 17.39% 13.98% 55.20% 0.00% : 15.51% ??? inst/repl : LDSTIssueX 0.06
 2 30000198 8.00% 18.00% 14.00% 54.67% 0.00% : 14.81% ??? inst/repl : LDSTIssueX 0.06
 3 30283347 8.15% 17.39% 13.98% 55.20% 0.00% : 15.51% ??? inst/repl : LDSTIssueX 0.06
 4 30000208 8.00% 18.00% 14.00% 54.67% 0.00% : 14.81% ??? inst/repl : LDSTIssueX 0.06
 5 30283351 8.15% 17.39% 13.98% 55.20% 0.00% : 15.51% ??? inst/repl : LDSTIssueX 0.06
 6 30000198 8.00% 18.00% 14.00% 54.67% 0.00% : 14.81% ??? inst/repl : LDSTIssueX 0.06
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
 1 0.35 90.00 86378760 35.1 0.0 0.0 59.7 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
 2 0.35 90.00 86379427 34.7 0.0 0.0 60.5 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0
 3 0.35 90.00 86378759 35.1 0.0 0.0 59.7 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
 4 0.35 90.00 86379429 34.7 0.0 0.0 60.5 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0
 5 0.35 90.00 86378758 35.1 0.0 0.0 59.7 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
 6 0.35 90.00 86379429 34.7 0.0 0.0 60.5 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0
#####
CACHE DE PETRIS
readHit      readMiss      writeHit      writeMiss
 0          0        2400000    7199968
#####

```

```

LOOP 500000
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestPrCo.o
# File : sesc_TestPrCo.o.Rm4ZPD : Tue Oct 18 01:43:33 2011
      Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
      786.319 KIPS    0.4173 MHz    574.990 secs   239.946 msec (rabbit)
      239946114.000
Proc Avg.Time BPType      Total      RAS      BPred      BTB      BTAC
  1 65.698 static    73.65% ( 97.92% of 32.43%) 62.00% (100.00% of 41.22%) 0.00%
  2 68.892 static    75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
  3 65.698 static    73.65% ( 97.92% of 32.43%) 62.00% (100.00% of 41.22%) 0.00%
  4 68.892 static    75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
  5 65.698 static    73.65% ( 97.92% of 32.43%) 62.00% (100.00% of 41.22%) 0.00%
  6 68.892 static    75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
      nInst     BJ      Load Store      INT      FP : LD Forward , Replay : Worst Unit (clk)
  1 75708343 8.15% 17.39% 13.98% 55.20% 0.00% : 15.51% ??? inst/repl : LDSTIssueX 0.06
  2 75000198 8.00% 18.00% 14.00% 54.67% 0.00% : 14.81% ??? inst/repl : LDSTIssueX 0.06
  3 75708347 8.15% 17.39% 13.98% 55.20% 0.00% : 15.51% ??? inst/repl : LDSTIssueX 0.06
  4 75000208 8.00% 18.00% 14.00% 54.67% 0.00% : 14.81% ??? inst/repl : LDSTIssueX 0.06
  5 75708351 8.15% 17.39% 13.98% 55.20% 0.00% : 15.51% ??? inst/repl : LDSTIssueX 0.06
  6 75000198 8.00% 18.00% 14.00% 54.67% 0.00% : 14.81% ??? inst/repl : LDSTIssueX 0.06
Proc IPC Active      Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
  1 0.35 90.00 215945421 35.1 0.0 0.0 59.7 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
  2 0.35 90.00 215946091 34.7 0.0 0.0 60.5 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0
  3 0.35 90.00 215945420 35.1 0.0 0.0 59.7 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
  4 0.35 90.00 215946093 34.7 0.0 0.0 60.5 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0
  5 0.35 90.00 215945422 35.1 0.0 0.0 59.7 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.0
  6 0.35 90.00 215946093 34.7 0.0 0.0 60.5 0.0 0.0 0.0 0.0 0.0 4.8 0.0 0.0
#####
CACHE DE PETRIS
readHit      readMiss      writeHit      writeMiss
  0          0       6000000     17999968
#####

```

### 12.2.3.2 Mediciones Obtenidas con el Simulador Original

```

LOOP 10000
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestPrCo-org.o
# File : sesc_TestPrCo-org.o.ZopIr4 : Tue Oct 18 01:48:25 2011
      Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
      816.134 KIPS    0.3922 MHz    10.050 secs   3.942 msec (rabbit)
      3941754.000
Proc Avg.Time BPType      Total      RAS      BPred      BTB      BTAC
  1 56.025 static    73.68% ( 97.97% of 32.45%) 62.00% ( 99.98% of 41.23%) 0.00%
  2 58.338 static    74.99% (100.00% of 33.32%) 62.50% ( 99.98% of 41.67%) 0.00%
  3 56.024 static    73.68% ( 97.97% of 32.45%) 62.00% ( 99.98% of 41.23%) 0.00%
  4 58.337 static    74.99% (100.00% of 33.32%) 62.50% ( 99.98% of 41.67%) 0.00%
  5 56.022 static    73.68% ( 97.97% of 32.45%) 62.00% ( 99.98% of 41.23%) 0.00%
  6 58.338 static    74.99% (100.00% of 33.32%) 62.50% ( 99.98% of 41.67%) 0.00%
      nInst     BJ      Load Store      INT      FP : LD Forward , Replay : Worst Unit (clk)
  1 1373816 8.97% 17.71% 12.49% 55.01% 0.00% : 8.56% ??? inst/repl : LDSTIssueX 0.04
  2 1360213 8.83% 18.38% 12.50% 54.41% 0.00% : 8.01% ??? inst/repl : LDSTIssueX 0.03
  3 1373810 8.97% 17.71% 12.49% 55.01% 0.00% : 8.56% ??? inst/repl : LDSTIssueX 0.04
  4 1360223 8.83% 18.38% 12.50% 54.41% 0.00% : 8.01% ??? inst/repl : LDSTIssueX 0.03
  5 1373871 8.97% 17.71% 12.49% 55.01% 0.00% : 8.56% ??? inst/repl : LDSTIssueX 0.04
  6 1360213 8.83% 18.38% 12.50% 54.41% 0.00% : 8.01% ??? inst/repl : LDSTIssueX 0.03
Proc IPC Active      Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
  1 0.35 99.98 3941117 34.9 0.0 0.0 59.4 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.1
  2 0.35 100.00 3941700 34.5 0.0 0.0 60.1 0.0 0.0 0.0 0.0 0.0 5.3 0.0 0.1
  3 0.35 99.98 3941084 34.9 0.0 0.0 59.4 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.1
  4 0.35 100.00 3941700 34.5 0.0 0.0 60.1 0.0 0.0 0.0 0.0 0.0 5.3 0.0 0.1
  5 0.35 99.99 3941312 34.9 0.0 0.0 59.4 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.1
  6 0.35 100.00 3941692 34.5 0.0 0.0 60.1 0.0 0.0 0.0 0.0 0.0 5.3 0.0 0.1
#####
CACHE DE PETRIS

```

```

readHit      readMiss      writeHit      writeMiss
 0          0          0          0
#####
#LOOP 15000
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestPrCo-org.o
# File : sesc_TestPrCo-org.o.6xa2Xm :      Tue Oct 18 01:49:43 2011
      Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
      807.802 KIPS  0.3882 MHz  15.230 secs  5.912 msec (rabbit)  5911755.000
Proc Avg.Time BPType      Total      RAS      BPred      BTB      BTAC
 1 56.020 static    73.68% ( 97.97% of 32.46%) 62.01% ( 99.99% of 41.23%) 0.00%
 2 58.336 static    75.00% (100.00% of 33.33%) 62.50% ( 99.99% of 41.67%) 0.00%
 3 56.018 static    73.68% ( 97.97% of 32.45%) 62.01% ( 99.99% of 41.23%) 0.00%
 4 58.336 static    75.00% (100.00% of 33.32%) 62.50% ( 99.99% of 41.67%) 0.00%
 5 56.018 static    73.68% ( 97.97% of 32.45%) 62.01% ( 99.99% of 41.23%) 0.00%
 6 58.336 static    75.00% (100.00% of 33.33%) 62.50% ( 99.99% of 41.67%) 0.00%
      nInst      BJ      Load      Store      INT      FP : LD Forward , Replay : Worst Unit (clk)
 1 2060701 8.97% 17.71% 12.49% 55.01% 0.00% : 8.56% ??? inst/repl : LDSTIssueX 0.04
 2 2040213 8.82% 18.38% 12.50% 54.41% 0.00% : 8.00% ??? inst/repl : LDSTIssueX 0.03
 3 2060712 8.97% 17.71% 12.49% 55.01% 0.00% : 8.56% ??? inst/repl : LDSTIssueX 0.04
 4 2040223 8.82% 18.38% 12.50% 54.41% 0.00% : 8.00% ??? inst/repl : LDSTIssueX 0.03
 5 2060756 8.97% 17.71% 12.49% 55.01% 0.00% : 8.56% ??? inst/repl : LDSTIssueX 0.04
 6 2040213 8.82% 18.38% 12.50% 54.41% 0.00% : 8.00% ??? inst/repl : LDSTIssueX 0.03
Proc IPC Active      Cycles      Busy      LDQ      STQ      IWin      ROB      Regs      Ports      TLB      maxBr      MisBr      Br4Clk      Other
 1 0.35 99.99 5911088 34.9 0.0 0.0 59.4 0.0 0.0 0.0 0.0 5.7 0.0 0.1
 2 0.35 100.00 5911701 34.5 0.0 0.0 60.1 0.0 0.0 0.0 0.0 5.3 0.0 0.1
 3 0.35 99.99 5911128 34.9 0.0 0.0 59.4 0.0 0.0 0.0 0.0 5.7 0.0 0.1
 4 0.35 100.00 5911702 34.5 0.0 0.0 60.1 0.0 0.0 0.0 0.0 5.3 0.0 0.1
 5 0.35 99.99 5911283 34.9 0.0 0.0 59.4 0.0 0.0 0.0 0.0 5.7 0.0 0.1
 6 0.35 100.00 5911692 34.5 0.0 0.0 60.1 0.0 0.0 0.0 0.0 5.3 0.0 0.1
#####
#CACHE DE PETRIS
readHit      readMiss      writeHit      writeMiss
 0          0          0          0
#####
#LOOP 20000
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestPrCo-org.o
# File : sesc_TestPrCo-org.o.3lHpfF :      Tue Oct 18 01:51:32 2011
      Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
      812.457 KIPS  0.3904 MHz  20.190 secs  7.882 msec (rabbit)  7881755.000
Proc Avg.Time BPType      Total      RAS      BPred      BTB      BTAC
 1 56.016 static    73.68% ( 97.97% of 32.46%) 62.01% ( 99.99% of 41.23%) 0.00%
 2 58.336 static    75.00% (100.00% of 33.33%) 62.50% ( 99.99% of 41.67%) 0.00%
 3 56.016 static    73.68% ( 97.97% of 32.46%) 62.01% ( 99.99% of 41.23%) 0.00%
 4 58.335 static    75.00% (100.00% of 33.33%) 62.50% ( 99.99% of 41.67%) 0.00%
 5 56.015 static    73.68% ( 97.97% of 32.45%) 62.01% ( 99.99% of 41.23%) 0.00%
 6 58.336 static    75.00% (100.00% of 33.33%) 62.50% ( 99.99% of 41.67%) 0.00%
      nInst      BJ      Load      Store      INT      FP : LD Forward , Replay : Worst Unit (clk)
 1 2747603 8.97% 17.71% 12.49% 55.01% 0.00% : 8.56% ??? inst/repl : LDSTIssueX 0.04
 2 2720213 8.82% 18.38% 12.50% 54.41% 0.00% : 8.00% ??? inst/repl : LDSTIssueX 0.03
 3 2747597 8.97% 17.71% 12.49% 55.01% 0.00% : 8.56% ??? inst/repl : LDSTIssueX 0.04
 4 2720223 8.82% 18.38% 12.50% 54.41% 0.00% : 8.00% ??? inst/repl : LDSTIssueX 0.03
 5 2747658 8.97% 17.71% 12.49% 55.01% 0.00% : 8.56% ??? inst/repl : LDSTIssueX 0.04
 6 2720213 8.82% 18.38% 12.50% 54.41% 0.00% : 8.00% ??? inst/repl : LDSTIssueX 0.03
Proc IPC Active      Cycles      Busy      LDQ      STQ      IWin      ROB      Regs      Ports      TLB      maxBr      MisBr      Br4Clk      Other
 1 0.35 99.99 7881131 34.9 0.0 0.0 59.4 0.0 0.0 0.0 0.0 5.7 0.0 0.1
 2 0.35 100.00 7881700 34.5 0.0 0.0 60.1 0.0 0.0 0.0 0.0 5.3 0.0 0.1
 3 0.35 99.99 7881098 34.9 0.0 0.0 59.4 0.0 0.0 0.0 0.0 5.7 0.0 0.1
 4 0.35 100.00 7881700 34.5 0.0 0.0 60.1 0.0 0.0 0.0 0.0 5.3 0.0 0.1
 5 0.35 99.99 7881326 34.9 0.0 0.0 59.4 0.0 0.0 0.0 0.0 5.7 0.0 0.1
 6 0.35 100.00 7881692 34.5 0.0 0.0 60.1 0.0 0.0 0.0 0.0 5.3 0.0 0.1

```

```
#####
CACHE DE PETRIS
readHit      readMiss      writeHit      writeMiss
 0          0          0          0
#####
```

## LOOP 30000

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestPrCo-org.o
# File : sesc_TestPrCo-org.o.blwvZl : Tue Oct 18 01:53:04 2011
      Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
      805.659 KIPS  0.3871 MHz  30.540 secs  11.822 msec (rabbit)  11821755.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
 1 56.014 static  73.68% (97.97% of 32.46%) 62.01% (99.99% of 41.23%) 0.00%
 2 58.335 static  75.00% (100.00% of 33.33%) 62.50% (99.99% of 41.67%) 0.00%
 3 56.013 static  73.68% (97.97% of 32.46%) 62.01% (99.99% of 41.23%) 0.00%
 4 58.335 static  75.00% (100.00% of 33.33%) 62.50% (99.99% of 41.67%) 0.00%
 5 56.013 static  73.68% (97.97% of 32.45%) 62.01% (99.99% of 41.23%) 0.00%
 6 58.335 static  75.00% (100.00% of 33.33%) 62.50% (99.99% of 41.67%) 0.00%
      nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
 1 4121373 8.97% 17.71% 12.49% 55.01% 0.00% : 8.56% ??? inst/repl : LDSTIssueX 0.04
 2 4080213 8.82% 18.38% 12.50% 54.41% 0.00% : 8.00% ??? inst/repl : LDSTIssueX 0.03
 3 4121384 8.97% 17.71% 12.49% 55.01% 0.00% : 8.56% ??? inst/repl : LDSTIssueX 0.04
 4 4080223 8.82% 18.38% 12.50% 54.41% 0.00% : 8.00% ??? inst/repl : LDSTIssueX 0.03
 5 4121428 8.97% 17.71% 12.49% 55.01% 0.00% : 8.56% ??? inst/repl : LDSTIssueX 0.04
 6 4080213 8.82% 18.38% 12.50% 54.41% 0.00% : 8.00% ??? inst/repl : LDSTIssueX 0.03
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
 1 0.35 99.99 11821072 34.9 0.0 0.0 59.4 0.0 0.0 0.0 0.0 5.7 0.0 0.1
 2 0.35 100.00 11821701 34.5 0.0 0.0 60.1 0.0 0.0 0.0 0.0 5.3 0.0 0.1
 3 0.35 99.99 11821111 34.9 0.0 0.0 59.4 0.0 0.0 0.0 0.0 5.7 0.0 0.1
 4 0.35 100.00 11821700 34.5 0.0 0.0 60.1 0.0 0.0 0.0 0.0 5.3 0.0 0.1
 5 0.35 100.00 11821267 34.9 0.0 0.0 59.4 0.0 0.0 0.0 0.0 5.7 0.0 0.1
 6 0.35 100.00 11821692 34.5 0.0 0.0 60.1 0.0 0.0 0.0 0.0 5.3 0.0 0.1
#####
CACHE DE PETRIS
readHit      readMiss      writeHit      writeMiss
 0          0          0          0
#####
```

## LOOP 40000

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestPrCo-org.o
# File : sesc_TestPrCo-org.o.URLj0J : Tue Oct 18 01:55:32 2011
      Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
      816.364 KIPS  0.3894 MHz  40.480 secs  15.762 msec (rabbit)  15761924.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
 1 54.469 static  73.68% (97.97% of 32.46%) 62.01% (100.00% of 41.23%) 0.00%
 2 56.668 static  75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
 3 54.469 static  73.68% (97.97% of 32.45%) 62.01% (100.00% of 41.23%) 0.00%
 4 56.668 static  75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
 5 54.469 static  73.68% (97.97% of 32.46%) 62.01% (100.00% of 41.23%) 0.00%
 6 56.668 static  75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
      nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
 1 5535212 8.91% 17.58% 12.40% 55.33% 0.00% : 8.56% ??? inst/repl : LDSTIssueX 0.04
 2 5480241 8.76% 18.25% 12.41% 54.74% 0.00% : 8.00% ??? inst/repl : LDSTIssueX 0.03
 3 5535263 8.91% 17.58% 12.40% 55.33% 0.00% : 8.56% ??? inst/repl : LDSTIssueX 0.04
 4 5480241 8.76% 18.25% 12.41% 54.74% 0.00% : 8.00% ??? inst/repl : LDSTIssueX 0.03
 5 5535210 8.91% 17.58% 12.40% 55.33% 0.00% : 8.56% ??? inst/repl : LDSTIssueX 0.04
 6 5480241 8.76% 18.25% 12.41% 54.74% 0.00% : 8.00% ??? inst/repl : LDSTIssueX 0.03
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
 1 0.35 100.00 15761359 35.1 0.0 0.0 59.1 0.0 0.0 0.0 0.0 5.7 0.0 0.1
 2 0.35 100.00 15761859 34.8 0.0 0.0 59.9 0.0 0.0 0.0 0.0 5.3 0.0 0.1
 3 0.35 100.00 15761492 35.1 0.0 0.0 59.1 0.0 0.0 0.0 0.0 5.7 0.0 0.1
 4 0.35 100.00 15761869 34.8 0.0 0.0 59.9 0.0 0.0 0.0 0.0 5.3 0.0 0.1
#####
```

```

5 0.35 100.00 15761350 35.1 0.0 0.0 59.1 0.0 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.1
6 0.35 100.00 15761870 34.8 0.0 0.0 59.9 0.0 0.0 0.0 0.0 0.0 0.0 5.3 0.0 0.1
#####
CACHE DE PETRIS
readHit      readMiss      writeHit      writeMiss
0           0           0           0
#####

LOOP 50000
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestPrCo-org.o
# File : sesc_TestPrCo-org.o.xALGF8 : Tue Oct 18 01:57:16 2011
      Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
      821.717 KIPS  0.3919 MHz  50.270 secs  19.702 msec (rabbit)  19701923.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
1 54.469 static    73.68% (97.97% of 32.46%) 62.01% (100.00% of 41.23%) 0.00%
2 56.668 static    75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
3 54.469 static    73.68% (97.97% of 32.45%) 62.01% (100.00% of 41.23%) 0.00%
4 56.668 static    75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
5 54.468 static    73.68% (97.97% of 32.46%) 62.01% (100.00% of 41.23%) 0.00%
6 56.668 static    75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
      nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
1   6918982 8.91% 17.58% 12.40% 55.33% 0.00% : 8.56% ??? inst/repl : LDSTIssueX 0.04
2   6850241 8.76% 18.25% 12.41% 54.74% 0.00% : 8.00% ??? inst/repl : LDSTIssueX 0.03
3   6919033 8.91% 17.58% 12.40% 55.33% 0.00% : 8.56% ??? inst/repl : LDSTIssueX 0.04
4   6850241 8.76% 18.25% 12.41% 54.74% 0.00% : 8.00% ??? inst/repl : LDSTIssueX 0.03
5   6918997 8.91% 17.58% 12.40% 55.33% 0.00% : 8.56% ??? inst/repl : LDSTIssueX 0.04
6   6850241 8.76% 18.25% 12.41% 54.74% 0.00% : 8.00% ??? inst/repl : LDSTIssueX 0.03
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.35 100.00 19701324 35.1 0.0 0.0 59.1 0.0 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.1
2 0.35 100.00 19701860 34.8 0.0 0.0 59.9 0.0 0.0 0.0 0.0 0.0 0.0 5.3 0.0 0.1
3 0.35 100.00 19701432 35.1 0.0 0.0 59.1 0.0 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.1
4 0.35 100.00 19701871 34.8 0.0 0.0 59.9 0.0 0.0 0.0 0.0 0.0 0.0 5.3 0.0 0.1
5 0.35 100.00 19701334 35.1 0.0 0.0 59.1 0.0 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.1
6 0.35 100.00 19701870 34.8 0.0 0.0 59.9 0.0 0.0 0.0 0.0 0.0 0.0 5.3 0.0 0.1
#####
CACHE DE PETRIS
readHit      readMiss      writeHit      writeMiss
0           0           0           0
#####

LOOP 100000
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestPrCo-org.o
# File : sesc_TestPrCo-org.o.8szXkt : Tue Oct 18 02:00:23 2011
      Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
      821.225 KIPS  0.3908 MHz  101.330 secs  39.603 msec (rabbit)  39602512.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
1 55.085 static    73.68% (97.97% of 32.46%) 62.01% (100.00% of 41.23%) 0.00%
2 57.334 static    75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
3 55.084 static    73.68% (97.97% of 32.46%) 62.01% (100.00% of 41.23%) 0.00%
4 57.334 static    75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
5 55.084 static    73.68% (97.97% of 32.46%) 62.01% (100.00% of 41.23%) 0.00%
6 57.334 static    75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
      nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
1   13937914 8.84% 17.45% 12.31% 55.65% 0.00% : 8.56% ??? inst/repl : LDSTIssueX
0.04
2   13800225 8.70% 18.12% 12.32% 55.07% 0.00% : 8.00% ??? inst/repl : LDSTIssueX
0.03
3   13937962 8.84% 17.45% 12.31% 55.65% 0.00% : 8.56% ??? inst/repl : LDSTIssueX
0.04
4   13800376 8.70% 18.12% 12.32% 55.07% 0.00% : 8.00% ??? inst/repl : LDSTIssueX
0.03
5   13937966 8.84% 17.45% 12.31% 55.65% 0.00% : 8.56% ??? inst/repl : LDSTIssueX

```

```

0.04
 6      13800272 8.70% 18.12% 12.32% 55.07% 0.00% :      8.00% ??? inst/repl : LDSTIssueX
0.03
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
 1 0.35 100.00 39601270 35.2 0.0 0.0 59.1 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.1
 2 0.35 100.00 39601852 34.8 0.0 0.0 59.8 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.1
 3 0.35 100.00 39601465 35.2 0.0 0.0 59.1 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.1
 4 0.35 100.00 39602458 34.8 0.0 0.0 59.8 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.1
 5 0.35 100.00 39601465 35.2 0.0 0.0 59.1 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.1
 6 0.35 100.00 39602066 34.8 0.0 0.0 59.8 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.1
#####
CACHE DE PETRIS
readHit      readMiss      writeHit      writeMiss
 0          0          0          0
#####

LOOP 200000
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestPrCo-org.o
# File : sesc_TestPrCo-org.o.on86Kx :      Tue Oct 18 02:05:06 2011
      Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
      815.185 KIPS  0.3879 MHz  204.160 secs  79.203 msec (rabbit)  79202511.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
 1 55.084 static 73.68% ( 97.97% of 32.46%) 62.01% (100.00% of 41.23%) 0.00%
 2 57.334 static 75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
 3 55.084 static 73.68% ( 97.97% of 32.46%) 62.01% (100.00% of 41.23%) 0.00%
 4 57.334 static 75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
 5 55.084 static 73.68% ( 97.97% of 32.46%) 62.01% (100.00% of 41.23%) 0.00%
 6 57.334 static 75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
      nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
 1 27875750 8.84% 17.45% 12.31% 55.65% 0.00% :      8.56% ??? inst/repl : LDSTIssueX
0.04
 2 27600225 8.70% 18.12% 12.32% 55.07% 0.00% :      8.00% ??? inst/repl : LDSTIssueX
0.03
 3 27875798 8.84% 17.45% 12.31% 55.65% 0.00% :      8.56% ??? inst/repl : LDSTIssueX
0.04
 4 27600376 8.70% 18.12% 12.32% 55.07% 0.00% :      8.00% ??? inst/repl : LDSTIssueX
0.03
 5 27875802 8.84% 17.45% 12.31% 55.65% 0.00% :      8.56% ??? inst/repl : LDSTIssueX
0.04
 6 27600272 8.70% 18.12% 12.32% 55.07% 0.00% :      8.00% ??? inst/repl : LDSTIssueX
0.03
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
 1 0.35 100.00 79201262 35.2 0.0 0.0 59.1 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.1
 2 0.35 100.00 79201854 34.8 0.0 0.0 59.8 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.1
 3 0.35 100.00 79201458 35.2 0.0 0.0 59.1 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.1
 4 0.35 100.00 79202457 34.8 0.0 0.0 59.8 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.1
 5 0.35 100.00 79201457 35.2 0.0 0.0 59.1 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.1
 6 0.35 100.00 79202066 34.8 0.0 0.0 59.8 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.1
#####
CACHE DE PETRIS
readHit      readMiss      writeHit      writeMiss
 0          0          0          0
#####

LOOP 500000
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf TestPrCo-org.o
# File : sesc_TestPrCo-org.o.yHQ3cF :      Tue Oct 18 02:14:08 2011
      Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
      835.060 KIPS  0.3974 MHz  498.250 secs  198.003 msec (rabbit)
      198002512.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
 1 55.084 static 73.68% ( 97.97% of 32.46%) 62.01% (100.00% of 41.23%) 0.00%

```

```

2 57.334 static      75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
3 55.084 static      73.68% (97.97% of 32.46%) 62.01% (100.00% of 41.23%) 0.00%
4 57.334 static      75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
5 55.084 static      73.68% (97.97% of 32.46%) 62.01% (100.00% of 41.23%) 0.00%
6 57.334 static      75.00% (100.00% of 33.33%) 62.50% (100.00% of 41.67%) 0.00%
    nInst   BJ      Load Store     INT     FP : LD Forward , Replay : Worst Unit (clk)
1   69689258 8.84% 17.45% 12.31% 55.65% 0.00% :           8.56% ??? inst/repl : LDSTIssueX
0.04
2   69000225 8.70% 18.12% 12.32% 55.07% 0.00% :           8.00% ??? inst/repl : LDSTIssueX
0.03
3   69689306 8.84% 17.45% 12.31% 55.65% 0.00% :           8.56% ??? inst/repl : LDSTIssueX
0.04
4   69000376 8.70% 18.12% 12.32% 55.07% 0.00% :           8.00% ??? inst/repl : LDSTIssueX
0.03
5   69689310 8.84% 17.45% 12.31% 55.65% 0.00% :           8.56% ??? inst/repl : LDSTIssueX
0.04
6   69000272 8.70% 18.12% 12.32% 55.07% 0.00% :           8.00% ??? inst/repl : LDSTIssueX
0.03
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.35 100.00 198001238 35.2 0.0 0.0 59.1 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.1
2 0.35 100.00 198001852 34.8 0.0 0.0 59.8 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.1
3 0.35 100.00 198001434 35.2 0.0 0.0 59.1 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.1
4 0.35 100.00 198002458 34.8 0.0 0.0 59.8 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.1
5 0.35 100.00 198001433 35.2 0.0 0.0 59.1 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.1
6 0.35 100.00 198002066 34.8 0.0 0.0 59.8 0.0 0.0 0.0 0.0 0.0 5.2 0.0 0.1
#####
##### CACHE DE PETRIS #####
readHit          readMiss        writeHit        writeMiss
0               0               0
#####

```

#### 12.2.4 CONTROL PLANTA DE EMBALAJE

##### CONTROL DE PLANTA DE EMBALAJE (para correr con simulador original)

```

#include "$HOME/esesc/src/libapp/sescapi.h"
#include "$HOME/esesc/src/libapp/sesc_locks.c"
#include "$HOME/esesc/src/libapp/sesc_events.c"

#define NUM_THREADS 2
#define LOOP 1000000
#define ALMACEN 5

ssema_t brazoRobot;      //Brazo Robotico
ssema_t almacenLibre;    //D
ssema_t almacenLleno;    //E

void work(void *threadid)
{
    int tid= sesc_self();

    int i=0,k=0;
    int *p;

    if(tid==1) {

```

```

for(int i=0; i<LOOP; i++) {

    printf("p1: la maquina M1 espera una pieza\n");
    //Toma brazo robotico
    sesc_psema(&brazoRobot);
    printf("p2: el robot carga la maquina M1 con una
pieza recibida desde la cinta A\n");
    //Libera brazo robotico
    sesc_vsema(&brazoRobot);

    printf("p3: la maquina M1 esta trabajando\n");
    printf("p4: la maquina M1 esta preparada para
descarga\n");
    //toma almacen y brazo robotico
    sesc_psema(&almacenLibre);
    sesc_psema(&brazoRobot);
    printf("p5: el robot descarga la maquina M1 y
almacena la pieza en el Almacen\n");
    //carga almacen, libera brazo
    sesc_vsema(&almacenLleno);
    sesc_vsema(&brazoRobot);
}
} else {

    for(int i=0; i<LOOP; i++) {

        printf("p9: la maquina M2 espera una pieza\n");
        //toma almacen y brazo robotico
        sesc_psema(&almacenLleno);
        sesc_psema(&brazoRobot);
        printf("p10: el robot carga la maquina M2 con una
pieza del Almacen\n");
        //descarga almacen y libera brazo
        sesc_vsema(&almacenLibre);
        sesc_vsema(&brazoRobot);
        printf("p6: la maquina M2 esta trabajando\n");
        printf("p7: la maquina M2 esta preparada para
descarga\n");
        //toma brazo robotico
        sesc_psema(&brazoRobot);
        printf("p8: el robot descarga la maquina M2 y envia
la pieza a la cinta B\n");
        //libera brazo robotico
        sesc_vsema(&brazoRobot);
    }
}

sesc_exit(0);

```

```

}

int main()
{
    int t;

    sesc_init();

    sesc_sema_init(&brazoRobot , (int32_t) 1);
    sesc_sema_init(&almacenLibre , (int32_t) ALMACEN);
    sesc_sema_init(&almacenLleno , (int32_t) 0);

    for(t=0;t<NUM_THREADS;t++) {
        sesc_spawn(work, (void
t,SESC_FLAG_NOMIGRATE|SESC_FLAG_MAP|t+1);
    }

    sesc_exit(0);
}

```

**CONTROL DE PLANTA DE EMBALAJE (para correr con simulador que implementa Petri)**

```

#include "$HOME/esesc/src/libapp/sescapi.h"
#include "$HOME/esesc/src/libapp/sesc_locks.c"
#include "$HOME/esesc/src/libapp/sesc_events.c"

#define NUM_THREADS 2
#define LOOP 1000000
#define ALMACEN 5

void work(void *threadid)
{
    int tid= sesc_self();

    int i=0,k=0;
    int *p;

    if(tid==1) {

        for(int i=0; i<LOOP; i++) {

            printf("p1: la maquina M1 espera una pieza\n");
            //DISPARO 0
            p = (int *) 405000000;
            *p=25;                                printf("p2: el robot carga la
maquina M1 con una pieza recibida desde la cinta A\n");
        }
    }
}

```

```

        //DISPARO 1
        p = (int *) 405000001;
        *p=25;

        printf("p3: la maquina M1 esta trabajando\n");
        //DISPARO 2
        p = (int *) 405000002;
        *p=25;
        printf("p4: la maquina M1 esta preparada para
descarga\n");
        //DISPARO 3
        p = (int *) 405000003;
        *p=25;
        printf("p5: el robot descarga la maquina M1 y
almacena la pieza en el Almacen\n");
        //DISPARO 4
        p = (int *) 405000004;
        *p=25;
    }
} else {

    for(int i=0; i<LOOP; i++) {

        printf("p9: la maquina M2 espera una pieza\n");
        //DISPARO 8
        p = (int *) 405000008;
        *p=25;
        printf("p10: el robot carga la maquina M2 con una
pieza del Almacen\n");
        //DISPARO 9
        p = (int *) 405000009;
        *p=25;
        printf("p6: la maquina M2 esta trabajando\n");
        //DISPARO 5
        p = (int *) 405000005;
        *p=25;
        printf("p7: la maquina M2 esta preparada para
descarga\n");
        //DISPARO 6
        p = (int *) 405000006;
        *p=25;
        printf("p8: el robot descarga la maquina M2 y envia
la pieza a la cinta B\n");
        //DISPARO 7
        p = (int *) 405000007;
        *p=25;      }
}

```

```

        sesc_exit(0);
    }

int main()
{
    int t;

    for(t=0;t<NUM_THREADS;t++) {
        sesc_spawn(work, (void
t,SESC_FLAG_NOMIGRATE|SESC_FLAG_MAP|t+1);
    }

    sesc_exit(0);
}

```

**CONTROL DE PLANTA DE EMBALAJE (Matriz de Incidencia y marcado inicial)**

M0 = [LOOP, 0, ALMACEN, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1]

I =	-1,0,0,0,0,0,0,0,0,0 0,0,0,0,0,0,0,1,0,0 0,0,0,-1,0,0,0,0,0,1 0,0,0,0,1,0,0,0,-1,0 -1,0,0,0,1,0,0,0,0,0 1,-1,0,0,0,0,0,0,0,0 0,1,-1,0,0,0,0,0,0,0 0,0,1,-1,0,0,0,0,0,0 0,0,0,1,-1,0,0,0,0,0 0,0,0,0,0,-1,0,0,0,1 0,0,0,0,0,1,-1,0,0,0 0,0,0,0,0,0,1,-1,0,0 0,0,0,0,0,0,1,-1,0 -1,1,0,-1,1,0,-1,1,-1,1
-----	--

**Archivo de configuración**

```

1
15,10
LOOP,0,ALMACEN,0,1,0,0,0,0,0,0,0,1,0,1
-1,0,0,0,0,0,0,0,0#0,0,0,0,0,0,0,1,0,0#0,0,0,-
1,0,0,0,0,0,1#0,0,0,0,1,0,0,-1,0#-1,0,0,0,1,0,0,0,0#1,-
1,0,0,0,0,0,0#0,1,-1,0,0,0,0,0#0,0,1,-1,0,0,0,0,0#0,0,0,1,-
1,0,0,0,0,0#0,0,0,0,-1,0,0,0,1#0,0,0,0,0,1,-1,0,0,0#0,0,0,0,0,1,-
1,0,0#0,0,0,0,0,0,1,-1,0#0,0,0,0,0,0,0,1,-1#-1,1,0,-1,1,0,-1,1,-
1,1
10
1,0,0,0,0,0,0,0,0#0,1,0,0,0,0,0,0#0,0,1,0,0,0,0,0#0,0,0,1,0,

```

```
0,0,0,0,0#0,0,0,1,0,0,0,0#0,0,0,0,1,0,0,0,0#0,0,0,0,1,0,0,0#
0,0,0,0,0,0,1,0,0#0,0,0,0,0,0,1,0#0,0,0,0,0,0,0,1
```

### 12.2.4.1 Mediciones Obtenidas con el Simulador Modificado

1000 productos a embalar

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf PlantaEmbalaje.o
# File : sesc_PlantaEmbalaje.o.03neel : Fri Oct 21 01:39:50 2011
    Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
    771.436 KIPS  1.1432 MHz  0.560 secs   0.640 msec (rabbit)  640213.000
                                         0.205 msec

Proc Avg.Time BPType      Total      RAS      BPred      BTB      BTAC
  1 48.853 static       63.87% ( 77.13% of 26.68%) 59.04% ( 99.89% of 37.23%) 0.00%
  2 49.669 static       65.56% ( 80.93% of 27.76%) 59.66% ( 99.87% of 37.77%) 0.00%
      nInst   BJ   Load Store   INT   FP : LD Forward , Replay : Worst Unit (clk)
  1     215345 10.44% 19.73% 16.14% 48.11% 0.00% : 22.06% ??? inst/repl : LDSTIssueX 0.08
  2     216659  9.98% 19.69% 16.30% 48.50% 0.00% : 21.48% ??? inst/repl : LDSTIssueX 0.08
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
  1 0.34 97.76 625864 34.4 0.0 0.0 56.4 0.0 0.0 0.0 0.0 9.1 0.0 0.0
  2 0.35 97.90 626788 34.6 0.0 0.0 57.0 0.0 0.0 0.0 0.0 8.4 0.0 0.1
#####
CACHE DE PETRIS
readHit      readMiss      writeHit      writeMiss
  0          0        10000      3373
#####
```

10000 productos a embalar

```
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf PlantaEmbalaje.o
# File : sesc_PlantaEmbalaje.o.NrC7Ls : Tue Oct 25 17:42:11 2011
    Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
    774.111 KIPS  1.1373 MHz  5.400 secs   6.142 msec (rabbit)  6141661.000
                                         1.792 msec

Proc Avg.Time BPType      Total      RAS      BPred      BTB      BTAC
  1 51.294 static       66.66% ( 83.33% of 28.57%) 59.99% ( 99.99% of 38.10%) 0.00%
  2 51.297 static       66.67% ( 83.34% of 28.57%) 60.00% ( 99.99% of 38.10%) 0.00%
      nInst   BJ   Load Store   INT   FP : LD Forward , Replay : Worst Unit (clk)
  1     2090021 10.05% 19.62% 16.27% 48.33% 0.00% : 21.95% ??? inst/repl : LDSTIssueX 0.08
  2     2090177 10.05% 19.62% 16.27% 48.32% 0.00% : 21.95% ??? inst/repl : LDSTIssueX 0.08
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
  1 0.35 97.06 5961193 35.1 0.0 0.0 56.7 0.0 0.0 0.0 0.0 8.2 0.0 0.0
  2 0.35 97.07 5961630 35.1 0.0 0.0 56.7 0.0 0.0 0.0 0.0 8.2 0.0 0.0
#####
CACHE DE PETRIS
readHit      readMiss      writeHit      writeMiss
  0          0        100000     79980
#####
```

100000 productos a embalar

```
Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf PlantaEmbalaje.o
# File : sesc_PlantaEmbalaje.o.uLutED : Tue Oct 25 17:51:56 2011
    Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
    786.068 KIPS  1.0859 MHz  49.360 secs  53.602 msec (rabbit)  53601944.000
                                         10.102 msec

Proc Avg.Time BPType      Total      RAS      BPred      BTB      BTAC
  1 68.500 static       76.47% (100.00% of 35.29%) 63.64% (100.00% of 41.18%) 0.00%
  2 68.251 static       76.47% (100.00% of 35.29%) 63.64% (100.00% of 41.18%) 0.00%
      nInst   BJ   Load Store   INT   FP : LD Forward , Replay : Worst Unit (clk)
  1     19400057 8.76% 19.07% 16.49% 49.48% 0.00% : 21.62% ??? inst/repl : LDSTIssueX 0.08
  2     19400247 8.76% 19.07% 16.49% 49.48% 0.00% : 21.62% ??? inst/repl : LDSTIssueX 0.08
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
  1 0.37 97.76 52401195 37.0 0.0 0.0 57.6 0.0 0.0 0.0 0.0 5.3 0.0 0.0
```

```

2 0.37 97.76 52401885 37.0 0.0 0.0 57.6 0.0 0.0 0.0 0.0 0.0 0.0 5.3 0.0 0.0
#####
CACHE DE PETRIS
readHit readMiss writeHit writeMiss
0 0 1000000 200008
#####

1000000 productos a embalar
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf PlantaEmbalaje.o
# File : sesc_PlantaEmbalaje.o.8fUkRd : Wed Oct 26 18:31:11 2011
      Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
    793.684 KIPS  1.0964 MHz  488.860 secs  536.002 msec (rabbit)  536001944.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
 1 68.500 static  76.47% (100.00% of 35.29%) 63.64% (100.00% of 41.18%) 0.00%
 2 68.250 static  76.47% (100.00% of 35.29%) 63.64% (100.00% of 41.18%) 0.00%
    nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
 1 194000057 8.76% 19.07% 16.49% 49.48% 0.00% : 21.62% ??? inst/repl : LDSTIssueX 0.08
 2 194000247 8.76% 19.07% 16.49% 49.48% 0.00% : 21.62% ??? inst/repl : LDSTIssueX 0.08
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
 1 0.37 97.76 524001195 37.0 0.0 0.0 57.6 0.0 0.0 0.0 0.0 0.0 0.0 5.3 0.0 0.0
 2 0.37 97.76 524001885 37.0 0.0 0.0 57.6 0.0 0.0 0.0 0.0 0.0 0.0 5.3 0.0 0.0
#####
CACHE DE PETRIS
readHit readMiss writeHit writeMiss
0 0 1000000 200008
#####

```

#### 12.2.4.2 Mediciones Obtenidas con el Simulador Original

```

1000 productos a embalar
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf PlantaEmbalaje-
org.o
# File : sesc_PlantaEmbalaje-org.o.h0qwLA : Fri Oct 21 01:40:35 2011
      Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
    812.552 KIPS  1.1124 MHz  0.420 secs  0.467 msec (rabbit)  467194.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
 1 52.411 static  71.40% ( 92.35% of 31.85%) 61.61% ( 99.88% of 39.60%) 0.00%
 2 56.181 static  73.59% ( 95.82% of 33.26%) 62.51% ( 99.85% of 40.30%) 0.00%
    nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
 1 169824 11.09% 19.93% 13.50% 48.42% 0.00% : 10.23% ??? inst/repl : LDSTIssueX 0.04
 2 171448 10.52% 19.87% 13.72% 48.89% 0.00% : 9.59% ??? inst/repl : LDSTIssueX 0.04
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
 1 0.37 99.54 465068 36.5 0.0 0.0 55.2 0.0 0.0 0.0 0.0 0.0 8.2 0.0 0.1
 2 0.37 99.99 467142 36.7 0.0 0.0 56.0 0.0 0.0 0.0 0.0 0.0 7.2 0.0 0.1
#####
CACHE DE PETRIS
readHit readMiss writeHit writeMiss
0 0 0 0
#####

```

```

10000 productos a embalar
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf PlantaEmbalaje-
org.o
# File : sesc_PlantaEmbalaje-org.o.61r7EX : Tue Oct 25 17:57:11 2011
      Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
    812.098 KIPS  1.0781 MHz  3.990 secs  4.302 msec (rabbit)  4301678.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
 1 61.503 static  76.46% (100.00% of 35.29%) 63.63% ( 99.99% of 41.18%) 0.00%
 2 61.507 static  76.47% (100.00% of 35.29%) 63.64% ( 99.98% of 41.18%) 0.00%
    nInst BJ Load Store INT FP : LD Forward , Replay : Worst Unit (clk)
 1 1620072 10.49% 19.75% 13.58% 48.77% 0.00% : 9.38% ??? inst/repl : LDSTIssueX 0.04
 2 1620198 10.49% 19.75% 13.58% 48.76% 0.00% : 9.38% ??? inst/repl : LDSTIssueX 0.04
Proc IPC Active Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other

```

```

1 0.38 99.99 4301237 37.7 0.0 0.0 55.8 0.0 0.0 0.0 0.0 0.0 6.5 0.0 0.1
2 0.38 100.00 4301627 37.7 0.0 0.0 55.8 0.0 0.0 0.0 0.0 0.0 6.5 0.0 0.1
#####
CACHE DE PETRIS
readHit    readMiss    writeHit    writeMiss
0          0          0          0
#####

100000 productos a embalar
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf PlantaEmbalaje-
org.o
# File : sesc_PlantaEmbalaje-org.o.4wU2wB : Tue Oct 25 17:55:55 2011
   Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
   812.292 KIPS 1.0699 MHz 40.380 secs 43.202 msec (rabbit) 43201874.000
Proc Avg.Time BPType Total     RAS      BPred     BTB      BTAC
1 60.751 static 76.47% (100.00% of 35.29%) 63.64% (100.00% of 41.18%) 0.00%
2 60.750 static 76.47% (100.00% of 35.29%) 63.64% (100.00% of 41.18%) 0.00%
   nInst   BJ   Load  Store   INT   FP : LD Forward , Replay : Worst Unit (clk)
1 16400097 10.37% 19.51% 13.41% 49.39% 0.00% : 9.38% ??? inst/repl : LDSTIssueX 0.04
2 16400251 10.37% 19.51% 13.41% 49.39% 0.00% : 9.38% ??? inst/repl : LDSTIssueX 0.04
Proc IPC Active   Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.38 100.00 43201328 38.0 0.0 0.0 55.6 0.0 0.0 0.0 0.0 6.4 0.0 0.1
2 0.38 100.00 43201822 38.0 0.0 0.0 55.6 0.0 0.0 0.0 0.0 6.4 0.0 0.1
#####
CACHE DE PETRIS
readHit    readMiss    writeHit    writeMiss
0          0          0          0
#####

1000000 productos a embalar
# Bench : /home/fede/sesc-build/sesc.smp -w1 -c /home/fede/esesc/confs/pseudoT1.conf PlantaEmbalaje-
org.o
# File : sesc_PlantaEmbalaje-org.o.oftSce : Wed Oct 26 17:44:59 2011
   Exe Speed   Exe MHz   Exe Time   Sim Time (1000MHz)   Cycles
   817.182 KIPS 1.0763 MHz 401.380 secs 432.002 msec (rabbit) 432001874.000
Proc Avg.Time BPType Total     RAS      BPred     BTB      BTAC
1 60.750 static 76.47% (100.00% of 35.29%) 63.64% (100.00% of 41.18%) 0.00%
2 60.750 static 76.47% (100.00% of 35.29%) 63.64% (100.00% of 41.18%) 0.00%
   nInst   BJ   Load  Store   INT   FP : LD Forward , Replay : Worst Unit (clk)
1 164000097 10.37% 19.51% 13.41% 49.39% 0.00% : 9.38% ??? inst/repl : LDSTIssueX 0.04
2 164000251 10.37% 19.51% 13.41% 49.39% 0.00% : 9.38% ??? inst/repl : LDSTIssueX 0.04
Proc IPC Active   Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.38 100.00 432001328 38.0 0.0 0.0 55.6 0.0 0.0 0.0 0.0 6.4 0.0 0.1
2 0.38 100.00 432001822 38.0 0.0 0.0 55.6 0.0 0.0 0.0 0.0 6.4 0.0 0.1
#####
CACHE DE PETRIS
readHit    readMiss    writeHit    writeMiss
0          0          0          0
#####

```

## 12.2.5 CONTROL DE CRUCERO

### CONTROL DE CRUCERO (para correr con simulador original)

```
#include "$HOME/esesc/src/libapp/sescapi.h"
#include "$HOME/esesc/src/libapp/sesc_locks.c"
#include "$HOME/esesc/src/libapp/sesc_events.c"

#define NUM_THREADS 4
#define LOOP 2

ssema_t semaS3;      // P3
ssema_t semaS4;      // P4
ssema_t semaS5;      // P2
ssema_t semaS6;      // P5

void work(void *threadid)
{
    int tid= sesc_self();
    int *p,i;

    if(tid==1){
        fprintf(stderr,"TURN ON \n");
        fprintf(stderr,"LEO VELOCITY \n");

        sesc_vsema(&semaS3);
        sesc_vsema(&semaS5);

    } else if (tid == 2) {
        // HILO PERSONA QUE MANEJA SUBIR/BAJAR VELOCIDAD Y FRENA

        for(i=0; i< LOOP; i++){
            sesc_psema(&semaS3);
            fprintf(stderr,"INCREMENT VELOCITY \n");
            sesc_vsema(&semaS3);

            sesc_psema(&semaS3);
            fprintf(stderr,"DECREMENT VELOCITY \n");
            sesc_vsema(&semaS3);

            sesc_psema(&semaS3);
            fprintf(stderr,"BRAKE \n");
            sesc_vsema(&semaS4);

            sesc_psema(&semaS4);
            sesc_psema(&semaS5);
            fprintf(stderr,"RESUME \n");
            sesc_vsema(&semaS3);
        }
    }
}
```

```

        sesc_vsema(&semas5);
    }

} else if (tid == 3) {
    for(i=0; i< LOOP; i++) {
        //Control de velocidad maxima
        sesc_psema(&semas4);
        fprintf(stderr,"CONTROL VELOCIDAD MAXIMA \n");
        sesc_vsema(&semas4);
    }
} else if (tid==4) {
    for(i=0; i< LOOP; i++) {
        sesc_psema(&semas3);
        sesc_psema(&semas5);
        fprintf(stderr,"DETECTA CERCANIA DE OBJETO / DESACELERA\n");
        sesc_vsema(&semas4);
        sesc_vsema(&semas6);

        fprintf(stderr,"SENSANDO CERCANIA DE OBJETOS\n");

        sesc_vsema(&semas5);
        sesc_psema(&semas6);

        //T10
        sesc_vsema(&semas6);
        sesc_psema(&semas5);

        fprintf(stderr,"SENSANDO CERCANIA DE OBJETOS - loop de sensado\n");

        sesc_vsema(&semas5);
        sesc_psema(&semas6);

        //T7
        sesc_psema(&semas4);
        sesc_psema(&semas5);
        fprintf(stderr,"RESUME \n");
        sesc_vsema(&semas3);
        sesc_vsema(&semas5);
    }
}

sesc_exit(0);
}

```

```

int main() {
    int t;
    sesc_init();

    sesc_sema_init(&semaS3 , (int32_t) 0);
    sesc_sema_init(&semaS4 , (int32_t) 0);
    sesc_sema_init(&semaS5 , (int32_t) 0);
    sesc_sema_init(&semaS6 , (int32_t) 0);

    for(t=0;t<NUM_THREADS;t++) {
        sesc_spawn(work, (void *) t, SESC_FLAG_NOMIGRATE|SESC_FLAG_MAP|t+1);
    }

    sesc_exit(0);
}

```

#### CONTROL DE CRUCERO (para correr con simulador que implementa Petri)

```

#include "$HOME/esesc/src/libapp/sescapi.h"
#include "$HOME/esesc/src/libapp/sesc_locks.c"
#include "$HOME/esesc/src/libapp/sesc_events.c"

#define NUM_THREADS 4
#define LOOP 2

void work(void *threadid)
{
    int tid= sesc_self();
    int *p,i;

    if(tid==1) {
        //T1 y T2
        p = (int *) 405000000;
        *p=25;
        fprintf(stderr,"TURN ON \n");
        p = (int *) 405000001;
        *p=25;
        fprintf(stderr,"LEO VELOCITY \n");
    } else if (tid == 2) {
        // HILO PERSONA QUE MANEJA SUBIR/BAJAR VELOCIDAD Y FRENA

        for(i=0; i< LOOP; i++) {
            //T3
            p = (int *) 405000002;

```

```

*p=25;
fprintf(stderr,"INCREMENT VELOCITY \n");

//T4
p = (int *) 405000003;
*p=25;

//T5
p = (int *) 405000004;
*p=25;
fprintf(stderr,"DECREMENT VELOCITY \n");

//T6
p = (int *) 405000005;
*p=25;

//T8
p = (int *) 405000007;
*p=25;
fprintf(stderr,"BRAKE \n");

//T7
p = (int *) 405000006;
*p=25;
fprintf(stderr,"RESUME \n");
}

} else if (tid == 3) {
    for(i=0; i< LOOP; i++) {
        //Control de velocidad maxima

        //T12
        p = (int *) 405000011;
        *p=25;
        fprintf(stderr,"CONTROL VELOCIDAD MAXIMA \n");

        //T13
        p = (int *) 405000012;
        *p=25;
    }

} else if (tid==4) {
    for(i=0; i< LOOP; i++) {
        //T9
        p = (int *) 405000008;
        *p=25;
        fprintf(stderr,"DETECTA CERCANIA DE OBJETO /"

```

```
DESACELERA\n");

    fprintf(stderr,"SENSANDO CERCANIA DE OBJETOS\n");

    //T11
    p = (int *) 405000010;
    *p=25;

    //T10
    p = (int *) 405000009;
    *p=25;

    fprintf(stderr,"SENSANDO CERCANIA DE OBJETOS - loop de
sensado\n");

    //T11
    p = (int *) 405000010;
    *p=25;

    //T7
    p = (int *) 405000006;
    *p=25;
    fprintf(stderr,"RESUME \n");
}

}

sesc_exit(0);

}

int main() {
    int t;

    for(t=0;t<NUM_THREADS;t++) {
        sesc_spawn(work,(void *)
t,SESC_FLAG_NOMIGRATE|SESC_FLAG_MAP|t+1);
    }

    sesc_exit(0);
}
```

### CONTROL DE CRUCERO (Matriz de Incidencia y marcado inicial)

```
M0 = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

I = | -1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 |
      | 1,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 |
      | 0,1,0,0,0,0,0,0,-1,-1,1,0,0,0,0,0,0,0,0,0 |
      | 0,1,-1,1,-1,1,1,-1,-1,0,0,0,0,0,0,0,0,0,0,0 |
      | 0,0,1,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 |
      | 0,0,0,0,1,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 |
      | 0,0,0,0,0,0,-1,1,1,-1,0,-1,1 |
      | 0,0,0,0,0,0,0,1,1,-1,0,0,0,0,0,0,0,0,0,0,0,0 |
      | 0,0,0,0,0,0,0,0,0,1,-1,-1,0,0,0,0,0,0,0,0,0,0 |
```

### Archivo de configuración

```
1
9,13
1,0,0,0,0,0,0,0,0
-1,0,0,0,0,0,0,0,0,0,0#1,-
1,0,0,0,0,0,0,0,0,0#0,1,0,0,0,0,0,-1,-1,1,0,0#0,1,-1,1,-
1,1,1,-1,-1,0,0,0#0,0,1,-1,0,0,0,0,0,0,0#0,0,0,0,1,-
1,0,0,0,0,0,0#0,0,0,0,0,-1,1,1,-1,0,-
1,1#0,0,0,0,0,0,0,1,1,-1,0,0#0,0,0,0,0,0,0,0,0,0,0,1,-1
13
1,0,0,0,0,0,0,0,0,0,0#0,1,0,0,0,0,0,0,0,0#0,0,1,0,0,0,0,
0,0,0,0,0,0#0,0,0,1,0,0,0,0,0,0,0#0,0,0,1,0,0,0,0,0,0,0,
0#0,0,0,0,0,1,0,0,0,0,0,0#0,0,0,0,0,1,0,0,0,0,0#0,0,0,0,
0,0,1,0,0,0,0#0,0,0,0,0,0,0,0,1,0,0,0#0,0,0,0,0,0,0,0,1,0,
0,0#0,0,0,0,0,0,0,0,0,1,0,0#0,0,0,0,0,0,0,0,0,0,0,1,0#0,0,0,
0,0,0,0,0,0,0,0,0,0,0,1,-
```

#### 12.2.5.1 Mediciones Obtenidas con el Simulador Modificado

```
LOOP=10
# Bench : /home/user/sesc-build/sesc.smp -w1 -c /home/user/esesc/confs/pseudoT1.conf ControlCrucero.o
# File : sesc_ControlCrucero.o.DmxBHg : Wed Oct 26 14:46:51 2011
          Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
          540.600 KIPS    0.9106 MHz    0.020 secs    0.018 msec (rabbit)    18213.000
Proc Avg.Time BPType   Total      RAS      BPred      BTB      BTAC
 1 129.000 static    28.57% (100.00% of 28.57%)  0.00% ( 0.00% of 42.86%)  0.00%
 2 68.028 static     85.46% (90.00% of 13.59%)  84.75% (95.50% of 45.24%)  0.00%
 3 70.128 static     71.11% (85.00% of 14.81%)  68.70% (86.67% of 44.44%)  0.00%
 4 64.850 static     72.51% (94.29% of 30.30%)  63.04% (88.89% of 40.91%)  0.00%
          nInst      BJ      Load Store      INT      FP : LD Forward , Replay : Worst Unit (clk)
 1      74 9.46% 14.86% 18.92% 51.35% 0.00% : 54.55% ??? inst/repl : LDSTIssueX 1.83
 2      5303 13.88% 27.66% 13.50% 41.18% 0.00% : 22.70% ??? inst/repl : LDSTIssueX 0.11
 3      1038 13.01% 24.28% 14.93% 43.93% 0.00% : 23.41% ??? inst/repl : LDSTIssueX 0.10
 4      4397 10.51% 20.81% 14.74% 47.58% 0.00% : 17.16% ??? inst/repl : LDSTIssueX 0.07
Proc IPC Active      Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
 1 0.09 4.34      791 9.4 0.0 0.0 13.0 5.8 0.0 0.0 0.0 71.6 0.0      0.2
```

```

2 0.30 98.40    17922 29.6 0.0 0.0 59.6 0.0 0.0 0.0 0.0 0.0 10.8 0.0 0.0
3 0.25 22.62    4119 25.2 0.0 0.0 49.0 0.0 0.0 0.0 0.0 0.0 25.7 0.0 0.1
4 0.33 74.20    13514 32.5 0.0 0.0 51.6 0.0 0.0 0.0 0.0 0.0 15.8 0.0 0.1
#####
CACHE DE PETRIS
readHit readMisswriteHit writeMiss
0      0     132     107
#####

LOOP=100
# Bench : /home/user/sesc-build/sesc.smp -w1 -c /home/user/esesc/confs/pseudoT1.conf ControlCrucero.o
# File : sesc_ControlCrucero.o.aA|NVH :           Wed Oct 26 14:49:04 2011
          Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
          503.714 KIPS  0.7877 MHz  0.210 secs   0.165 msec (rabbit)  165424.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
1 129.000 static  28.57% (100.00% of 28.57%) 0.00% ( 0.00% of 42.86%) 0.00%
2 63.721 static   87.10% ( 90.00% of 14.07%) 86.62% ( 99.53% of 45.07%) 0.00%
3 61.049 static   75.29% ( 81.00% of 15.26%) 74.26% ( 98.59% of 43.33%) 0.00%
4 62.020 static   76.42% ( 93.79% of 30.45%) 68.82% ( 98.87% of 40.52%) 0.00%
nInst      BJ      Load Store      INT      FP : LD Forward , Replay : Worst Unit (clk)
1      74 9.46% 14.86% 18.92% 51.35% 0.00% : 54.55% ??? inst/repl : LDSTIssueX 1.83
2      51733 13.74% 27.46% 13.54% 41.39% 0.00% : 22.55% ??? inst/repl : LDSTIssueX 0.11
3      10143 12.93% 24.29% 14.87% 43.97% 0.00% : 23.01% ??? inst/repl : LDSTIssueX 0.10
4      43830 10.49% 20.86% 14.72% 47.55% 0.00% : 17.08% ??? inst/repl : LDSTIssueX 0.07
Proc IPC Active      Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.09 0.48      791 9.4 0.0 0.0 13.0 5.8 0.0 0.0 0.0 0.0 71.6 0.0 0.2
2 0.32 98.42     162809 31.8 0.0 0.0 63.6 0.0 0.0 0.0 0.0 0.0 4.7 0.0 0.0
3 0.31 19.86     32846 30.9 0.0 0.0 59.8 0.0 0.0 0.0 0.0 0.0 9.2 0.0 0.0
4 0.36 73.89     122228 35.9 0.0 0.0 56.9 0.0 0.0 0.0 0.0 0.0 7.2 0.0 0.0
#####
CACHE DE PETRIS
readHit readMisswriteHit writeMiss
0      0     1302     1260
#####

LOOP=1000
# Bench : /home/user/sesc-build/sesc.smp -w1 -c /home/user/esesc/confs/pseudoT1.conf ControlCrucero.o
# File : sesc_ControlCrucero.o.SC5NjC :           Wed Oct 26 14:49:52 2011
          Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
          522.853 KIPS  0.8105 MHz  2.020 secs   1.637 msec (rabbit)  1637273.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
1 129.000 static  28.57% (100.00% of 28.57%) 0.00% ( 0.00% of 42.86%) 0.00%
2 63.216 static   87.26% ( 90.00% of 14.13%) 86.81% ( 99.95% of 45.06%) 0.00%
3 59.962 static   75.90% ( 80.25% of 15.09%) 75.12% ( 99.86% of 43.25%) 0.00%
4 61.754 static   76.84% ( 93.77% of 30.47%) 69.42% ( 99.89% of 40.49%) 0.00%
nInst      BJ      Load Store      INT      FP : LD Forward , Replay : Worst Unit (clk)
1      74 9.46% 14.86% 18.92% 51.35% 0.00% : 54.55% ??? inst/repl : LDSTIssueX 1.83
2      515913 13.72% 27.44% 13.55% 41.42% 0.00% : 22.53% ??? inst/repl : LDSTIssueX 0.11
3      102092 12.98% 24.41% 14.82% 43.87% 0.00% : 23.00% ??? inst/repl : LDSTIssueX 0.10
4      438085 10.49% 20.86% 14.72% 47.55% 0.00% : 17.07% ??? inst/repl : LDSTIssueX 0.07
Proc IPC Active      Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.09 0.05      791 9.4 0.0 0.0 13.0 5.8 0.0 0.0 0.0 0.0 71.6 0.0 0.2
2 0.32 98.42     1611384 32.0 0.0 0.0 64.0 0.0 0.0 0.0 0.0 0.0 4.0 0.0 0.0
3 0.32 19.76     323492 31.6 0.0 0.0 61.3 0.0 0.0 0.0 0.0 0.0 7.1 0.0 0.0
4 0.36 73.84     1208988 36.2 0.0 0.0 57.5 0.0 0.0 0.0 0.0 0.0 6.2 0.0 0.0
#####
CACHE DE PETRIS
readHit readMisswriteHit writeMiss
0      0     13002     12834
#####

```

```

LOOP=10000
# Bench : /home/user/sesc-build/sesc.smp -w1 -c /home/user/esesc/confs/pseudoT1.conf ControlCrucero.o
# File : sesc_ControlCrucero.o.Lchbwe :      Wed Oct 26 14:50:46 2011
          Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
          513.835 KIPS  0.7959 MHz  20.550 secs  16.355 msec (rabbit)  16354983.000
Proc Avg.Time BPType  Total      RAS      BPred      BTB      BTAC
1 129.000 static    28.57% (100.00% of 28.57%)  0.00% ( 0.00% of 42.86%)  0.00%
2 63.165 static    87.28% (90.00% of 14.13%)  86.83% (100.00% of 45.05%)  0.00%
3 59.857 static    75.95% (80.20% of 15.08%)  75.20% ( 99.99% of 43.24%)  0.00%
4 61.733 static    76.89% (93.77% of 30.48%)  69.48% ( 99.99% of 40.48%)  0.00%
nInst      BJ      Load Store      INT      FP : LD Forward , Replay : Worst Unit (clk)
1       74 9.46% 14.86% 18.92% 51.35% 0.00% : 54.55% ??? inst/repl : LDSTIssueX 1.83
2      5157513 13.72% 27.44% 13.55% 41.42% 0.00% : 22.53% ??? inst/repl : LDSTIssueX 0.11
3     1021254 12.99% 24.42% 14.82% 43.86% 0.00% : 23.00% ??? inst/repl : LDSTIssueX 0.10
4     4380473 10.49% 20.86% 14.72% 47.55% 0.00% : 17.07% ??? inst/repl : LDSTIssueX 0.07
Proc IPC Active      Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.09 0.00      791 9.4 0.0 0.0 13.0 5.8 0.0 0.0 0.0 0.0 71.6 0.0      0.2
2 0.32 98.42    16096338 32.0 0.0 0.0 64.0 0.0 0.0 0.0 0.0 0.0 3.9 0.0 0.0
3 0.32 19.74    3228691 31.6 0.0 0.0 61.5 0.0 0.0 0.0 0.0 0.0 6.9 0.0 0.0
4 0.36 73.84    12075941 36.3 0.0 0.0 57.6 0.0 0.0 0.0 0.0 0.0 6.1 0.0 0.0
#####
CACHE DE PETRIS
readHit  readMisswriteHit writeMiss
0       0      130002      128591
#####

LOOP=100000
# Bench : /home/user/sesc-build/sesc.smp -w1 -c /home/user/esesc/confs/pseudoT1.conf ControlCrucero.o
# File : sesc_ControlCrucero.o.Zfnxe2 :      Wed Oct 26 14:56:38 2011
          Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
          518.331 KIPS  0.7988 MHz  201.740 secs  161.154 msec (rabbit)  161154338.000
Proc Avg.Time BPType  Total      RAS      BPred      BTB      BTAC
1 129.000 static    28.57% (100.00% of 28.57%)  0.00% ( 0.00% of 42.86%)  0.00%
2 62.858 static    87.04% (90.00% of 14.40%)  86.54% (100.00% of 44.96%)  0.00%
3 59.755 static    74.91% (80.44% of 15.81%)  73.88% (100.00% of 42.96%)  0.00%
4 64.951 static    78.03% (95.81% of 31.51%)  69.86% (100.00% of 40.80%)  0.00%
nInst      BJ      Load Store      INT      FP : LD Forward , Replay : Worst Unit (clk)
1       74 9.46% 14.86% 18.92% 51.35% 0.00% : 54.55% ??? inst/repl : LDSTIssueX 1.83
2      51117625 13.58% 27.17% 13.54% 41.80% 0.00% : 22.48% ??? inst/repl : LDSTIssueX 0.11
3     10108909 12.52% 23.48% 14.67% 45.38% 0.00% : 22.89% ??? inst/repl : LDSTIssueX 0.10
4     43341389 10.25% 20.66% 14.70% 47.93% 0.00% : 16.90% ??? inst/repl : LDSTIssueX 0.07
Proc IPC Active      Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.09 0.00      792 9.3 0.0 0.0 13.0 5.8 0.0 0.0 0.0 0.0 71.7 0.0      0.2
2 0.32 98.40    158580392 32.2 0.0 0.0 63.8 0.0 0.0 0.0 0.0 0.0 4.0 0.0 0.0
3 0.32 19.46    31358158 32.2 0.0 0.0 60.7 0.0 0.0 0.0 0.0 0.0 7.0 0.0 0.0
4 0.37 73.40    118280025 36.6 0.0 0.0 57.6 0.0 0.0 0.0 0.0 0.0 5.7 0.0 0.0
#####
CACHE DE PETRIS
readHit  readMisswriteHit writeMiss
0       0      130002      1273891
#####

```

### 12.2.5.2 Mediciones Obtenidas con el Simulador Original

```

LOOP = 10
# Bench : /home/user/sesc-build/sesc.smp -w1 -c /home/user/esesc/confs/pseudoT1.conf ControlCrucero.o
# File : sesc_ControlCrucero.o.lsbn91 :      Wed Oct 26 15:13:12 2011
          Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
          503.150 KIPS  0.8011 MHz  0.020 secs   0.016 msec (rabbit)  16022.000
Proc Avg.Time BPType  Total      RAS      BPred      BTB      BTAC
1 121.600 static    28.57% (100.00% of 28.57%)  0.00% ( 0.00% of 42.86%)  0.00%
2 62.271 static    84.80% (90.00% of 14.20%)  83.94% ( 95.27% of 45.03%)  0.00%

```

```

3 60.143 static      75.44% ( 80.00% of 11.70%) 74.83% ( 89.61% of 45.03%) 0.00%
4 63.223 static      74.03% ( 95.71% of 30.04%) 64.72% ( 89.12% of 41.42%) 0.00%
  nInst      BJ      Load Store      INT      FP : LD Forward , Replay : Worst Unit (clk)
1       62 11.29% 14.52% 16.13% 51.61% 0.00% : 44.44% ??? inst/repl : LDSTIssueX 2.01
2      4783 14.72% 28.08% 12.13% 40.89% 0.00% : 19.14% ??? inst/repl : LDSTIssueX 0.10
3     1095 15.62% 27.40% 12.15% 41.19% 0.00% : 18.67% ??? inst/repl : LDSTIssueX 0.10
4     4123 11.30% 21.37% 13.34% 47.20% 0.00% : 12.60% ??? inst/repl : LDSTIssueX 0.06
Proc IPC Active      Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.08 4.69      751 8.3 0.0 0.0 11.2 6.1 0.0 0.0 0.0 0.0 74.3 0.0      0.2
2 0.30 99.67    15969 30.0 0.0 0.0 59.5 0.0 0.0 0.0 0.0 0.0 10.5 0.0      0.0
3 0.26 26.54     4252 25.8 0.0 0.0 52.4 0.0 0.0 0.0 0.0 0.0 21.8 0.0      0.1
4 0.33 78.59    12592 32.7 0.0 0.0 51.1 0.0 0.0 0.0 0.0 0.0 16.1 0.0      0.1
#####
CACHE DE PETRIS
readHit  readMisswriteHit writeMiss
0      0      0      0
#####

LOOP = 100
# Bench : /home/user/sesc-build/sesc.smp -w1 -c /home/user/esesc/confs/pseudoT1.conf ControlCrucero.o
# File : sesc_ControlCrucero.o.Nz0f1m :      Wed Oct 26 15:14:10 2011
      Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
      541.056 KIPS  0.8180 MHz  0.180 secs  0.147 msec (rabbit)  147238.000
Proc Avg.Time BPType      Total      RAS      BPred      BTB      BTAC
1 121.600 static      28.57% (100.00% of 28.57%) 0.00% ( 0.00% of 42.86%) 0.00%
2 60.069 static      86.65% ( 90.00% of 14.56%) 86.08% ( 99.51% of 44.91%) 0.00%
3 55.054 static      77.07% ( 80.00% of 13.90%) 76.59% ( 98.73% of 43.78%) 0.00%
4 58.752 static      76.77% ( 94.21% of 30.51%) 69.10% ( 98.87% of 40.63%) 0.00%
  nInst      BJ      Load Store      INT      FP : LD Forward , Replay : Worst Unit (clk)
1       62 11.29% 14.52% 16.13% 51.61% 0.00% : 44.44% ??? inst/repl : LDSTIssueX 2.01
2      46953 14.63% 27.97% 12.11% 41.02% 0.00% : 18.92% ??? inst/repl : LDSTIssueX 0.10
3      9577 15.03% 26.23% 12.24% 42.33% 0.00% : 17.08% ??? inst/repl : LDSTIssueX 0.09
4     40798 11.25% 21.19% 13.35% 47.35% 0.00% : 12.28% ??? inst/repl : LDSTIssueX 0.05
Proc IPC Active      Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.08 0.51      751 8.3 0.0 0.0 11.2 6.1 0.0 0.0 0.0 0.0 74.3 0.0      0.2
2 0.32 99.96    147185 31.9 0.0 0.0 63.1 0.0 0.0 0.0 0.0 0.0 5.0 0.0 0.0
3 0.31 21.30    31355 30.5 0.0 0.0 60.1 0.0 0.0 0.0 0.0 0.0 9.3 0.0 0.1
4 0.36 76.57    112745 36.2 0.0 0.0 56.1 0.0 0.0 0.0 0.0 0.0 7.6 0.0 0.0
#####
CACHE DE PETRIS
readHit  readMisswriteHit writeMiss
0      0      0      0
#####

LOOP = 1000
# Bench : /home/user/sesc-build/sesc.smp -w1 -c /home/user/esesc/confs/pseudoT1.conf ControlCrucero.o
# File : sesc_ControlCrucero.o.pU92o1 :      Wed Oct 26 15:14:54 2011
      Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
      536.119 KIPS  0.8054 MHz  1.810 secs  1.458 msec (rabbit)  1457771.000
Proc Avg.Time BPType      Total      RAS      BPred      BTB      BTAC
1 121.600 static      28.57% (100.00% of 28.57%) 0.00% ( 0.00% of 42.86%) 0.00%
2 59.807 static      86.83% ( 90.00% of 14.61%) 86.28% ( 99.95% of 44.89%) 0.00%
3 54.379 static      77.33% ( 79.75% of 14.06%) 76.93% ( 99.87% of 43.64%) 0.00%
4 58.417 static      77.11% ( 94.19% of 30.62%) 69.58% ( 99.89% of 40.57%) 0.00%
  nInst      BJ      Load Store      INT      FP : LD Forward , Replay : Worst Unit (clk)
1       62 11.29% 14.52% 16.13% 51.61% 0.00% : 44.44% ??? inst/repl : LDSTIssueX 2.01
2      468213 14.62% 27.95% 12.11% 41.04% 0.00% : 18.89% ??? inst/repl : LDSTIssueX 0.10
3      94902 14.99% 26.15% 12.24% 42.41% 0.00% : 16.95% ??? inst/repl : LDSTIssueX 0.09
4     407199 11.23% 21.17% 13.35% 47.38% 0.00% : 12.24% ??? inst/repl : LDSTIssueX 0.05
Proc IPC Active      Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.08 0.05      751 8.3 0.0 0.0 11.2 6.1 0.0 0.0 0.0 0.0 74.3 0.0      0.2
2 0.32 100.00   1457717 32.1 0.0 0.0 63.5 0.0 0.0 0.0 0.0 0.0 4.4 0.0 0.0
#####

```

```

3 0.31 20.87    304309 31.2 0.0 0.0 61.2 0.0 0.0 0.0 0.0 0.0 7.6 0.0 0.0
4 0.37 76.33    1112784 36.6 0.0 0.0 56.7 0.0 0.0 0.0 0.0 0.0 6.6 0.0 0.0
#####
CACHE DE PETRIS
readHit readMisswriteHit writeMiss
0      0      0      0
#####

LOOP = 10000
# Bench : /home/user/sesc-build/sesc.smp -w1 -c /home/user/esesc/confs/pseudoT1.conf ControlCrucero.o
# File : sesc_ControlCrucero.o.7w0VA3 :   Wed Oct 26 15:15:41 2011
          Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
          534.729 KIPS  0.8053 MHz  18.120 secs  14.592 msec (rabbit)  14591617.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
1 121.600 static  28.57% (100.00% of 28.57%) 0.00% ( 0.00% of 42.86%) 0.00%
2 59.781 static   86.87% ( 90.00% of 14.58%) 86.34% (100.00% of 44.90%) 0.00%
3 54.342 static   76.87% ( 80.58% of 14.61%) 76.24% ( 99.99% of 43.51%) 0.00%
4 57.932 static   76.98% ( 93.91% of 30.52%) 69.54% ( 99.99% of 40.51%) 0.00%
nInst      BJ      Load Store      INT      FP : LD Forward , Replay : Worst Unit (clk)
1       62 11.29% 14.52% 16.13% 51.61% 0.00% : 44.44% ??? inst/repl : LDSTIssueX 2.01
2     4688523 14.63% 27.97% 12.11% 41.03% 0.00% : 18.90% ??? inst/repl : LDSTIssueX 0.10
3     922962 14.84% 25.82% 12.29% 42.72% 0.00% : 16.61% ??? inst/repl : LDSTIssueX 0.09
4     4077750 11.25% 21.17% 13.35% 47.37% 0.00% : 12.26% ??? inst/repl : LDSTIssueX 0.05
Proc IPC Active      Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.08 0.01      751 8.3 0.0 0.0 11.2 6.1 0.0 0.0 0.0 0.0 74.3 0.0      0.2
2 0.32 100.00    14591563 32.1 0.0 0.0 63.5 0.0 0.0 0.0 0.0 0.0 4.3 0.0      0.0
3 0.31 20.14     2938759 31.4 0.0 0.0 61.0 0.0 0.0 0.0 0.0 0.0 7.5 0.0 0.0
4 0.37 76.36     11141638 36.6 0.0 0.0 56.8 0.0 0.0 0.0 0.0 0.0 6.6 0.0 0.0
#####
CACHE DE PETRIS
readHit readMisswriteHit writeMiss
0      0      0      0
#####

LOOP = 100000
# Bench : /home/user/sesc-build/sesc.smp -w1 -c /home/user/esesc/confs/pseudoT1.conf ControlCrucero.o
# File : sesc_ControlCrucero.o.G5IWLN :   Wed Oct 26 15:19:30 2011
          Exe Speed      Exe MHz      Exe Time      Sim Time (1000MHz)      Cycles
          532.630 KIPS  0.7948 MHz  183.030 secs  145.472 msec (rabbit)  145471980.000
Proc Avg.Time BPType Total RAS BPred BTB BTAC
1 121.600 static  28.57% (100.00% of 28.57%) 0.00% ( 0.00% of 42.86%) 0.00%
2 59.445 static   86.81% ( 90.00% of 14.66%) 86.26% (100.00% of 44.87%) 0.00%
3 53.362 static   77.44% ( 79.26% of 13.90%) 77.15% (100.00% of 43.64%) 0.00%
4 58.631 static   77.35% ( 94.52% of 30.79%) 69.71% (100.00% of 40.62%) 0.00%
nInst      BJ      Load Store      INT      FP : LD Forward , Replay : Worst Unit (clk)
1       62 11.29% 14.52% 16.13% 51.61% 0.00% : 44.44% ??? inst/repl : LDSTIssueX 2.01
2     46913745 14.54% 27.81% 12.07% 41.32% 0.00% : 18.87% ??? inst/repl : LDSTIssueX 0.10
3     9768032 14.73% 25.71% 11.97% 43.50% 0.00% : 17.04% ??? inst/repl : LDSTIssueX 0.09
4     40805355 11.14% 21.05% 13.29% 47.66% 0.00% : 12.19% ??? inst/repl : LDSTIssueX 0.05
Proc IPC Active      Cycles Busy LDQ STQ IWin ROB Regs Ports TLB maxBr MisBr Br4Clk Other
1 0.08 0.00      751 8.3 0.0 0.0 11.2 6.1 0.0 0.0 0.0 0.0 74.3 0.0      0.2
2 0.32 100.00    145471927 32.2 0.0 0.0 63.4 0.0 0.0 0.0 0.0 0.0 4.3 0.0      0.0
3 0.32 21.22     30863002 31.6 0.0 0.0 61.0 0.0 0.0 0.0 0.0 0.0 7.3 0.0 0.0
4 0.37 76.22     110872016 36.8 0.0 0.0 56.7 0.0 0.0 0.0 0.0 0.0 6.5 0.0 0.0
#####
CACHE DE PETRIS
readHit readMisswriteHit writeMiss
0      0      0      0
#####

```



## 13 ACRÓNIMOS

---

ACRÓNIMO	SIGNIFICADO ESPAÑOL	SIGNIFICADO INGLES
<b>ALU</b>	Unidad Lógica Aritmética	Arithmetic Unit Logic
<b>API</b>	Interfaz de Programación de Aplicaciones	Application Programming Interface
<b>CMP</b>	Procesador de Memoria Común	Common Memory Processor
<b>CPU</b>	Unidad de Procesamiento Central	Central processing unit
<b>DES</b>	Sistemas de eventos discretos	Discrete-event systems
<b>FIFO</b>	Primero en ingresar, primero en salir.	First input, first output
<b>FPGA</b>	dispositivos lógicos de propósito general programable	Field Programmable Gate Array
<b>IDE</b>	Entorno de Desarrollo Integrado	integrated development environment
<b>ISA</b>	Arquitectura del Set de Instrucciones	Instruction Set Architecture
<b>MINT</b>	Interprete de Mips	Mips Interpreter
<b>MIPS</b>	Microprocesador sin Interbloqueo en las etapas del Pipeline	Microprocessor without Interlocked Pipeline Stages
<b>MMD</b>	Múltiple Instrucción Múltiples Datos	Multiple Instruction Multiple Data
<b>MS</b>	Milisegundos	
<b>PIM</b>	Procesamiento en Memoria	Processing In Memory
<b>PN</b>	Red de Petri	Petri Net
<b>RAM</b>	Memoria de Acceso aleatorio	Random Access Memory

ACRÓNIMO	SIGNIFICADO ESPAÑOL	SIGNIFICADO INGLES
<b>RCC</b>	Región Crítica Condicional	
<b>ROB</b>	Buffer de Reordenamiento	Reordering Buffer
<b>SESC</b>	Simulador Súper Escalar	Super Scalar Simulator
<b>SMP</b>	Multiprocesador Simétrico	Symmetric Multiprocessor
<b>SO</b>	Sistema Operativo	Operative System
<b>TLS</b>	Especulación a Nivel de Hilos	Thread Level Speculation

## 14 BIBLIOGRAFÍA

---

1. **Pailer, Julio.** *Medición del costo de sincronización, con carga de variables de cache y tiempos de ejecución de procesos concurrentes en un procesador multicore.* Córdoba : FCEFyN - UNC, 2010.
2. **Wolfgang, Mauerer.** *Professional Linux Kernel Architecture.* Indiana : Wiley Publishing, Inc., 2008.
3. **Diaz, Michel.** *Petri Nets: Fundamental Models, Verification and Applications.* s.l. : Wiley, 2009.
4. **Shannon, Robert y Johannes, James D.** *Systems Simulation: The Art and Science.* s.l. : IEEE Systems, Man, and Cybernetics Society, 1976. 0018-9472.
5. **Montesinos Ortego, Pablo y Sack, Paul.** The i-acoma group at University of Illinois at Urbana-Champaign. [En línea] <http://iacoma.cs.uiuc.edu/~paulsack/sescdoc/>.
6. **Iordache, Marian V. y Antsaklis, Panos J.** *Supervisory Control of Concurrent System: A Petri Net Structural Approach.* s.l. : Birkhauser, 2006.
7. **Pailer, Julio y Tejeda, Miguel.** *Implementación en hardware de un procesador de Petri para resolver problemas de sincronización y exclusión mutua en sistemas multicore.* s.l. : FCEFyN-UNC, 2011.
8. **Palma Méndez, José Tomás y Garrido Carrera, María del Carmen.** *Programación Concurrente.* Madrid, España : Thomson, 2003.
9. **Bucci, G., y otros, y otros.** *Modeling Flexible Real Time Systems with Preemptive Time Petri Nets.* Firenze : Universita di Firenze.
10. **Hopcroft, J. E. y Ullman, J. D.** *Formal Languages and their Relation to Automata.* s.l. : Addison-Wesley, 1969.
11. **Jantzen, Matthias.** *On the hierarchy of Petri net languages.* s.l. : AFCET, 1979.
12. **Buhr, Peter y Fortier, Michel.** *Monitor Classification.* s.l. : ACM Computing Surveys, 1995.
13. **Andrews, G. R. y Schneider, F. B.** *Concepts and notations for concurrent programing.* s.l. : ACM Cornput, 1983.
14. **Gray, John Shapley.** *Interprocess Communications in Linux.* s.l. : Prentice Hall PTR, 2003.
15. **Castellanos Arias, Johanna Stella.** *Modeling with Petri Nets and Implementation with Grafset of a Flexible Manufacturing System with Concurrent Processes and Shared Resources.*

16. **von Prof. Dr. Gold, Robert.** *Petri Nets in Software Engineering.* s.l. : Ingolstadt, 2004. ISSN 1612-6483 .
17. **Patterson, David y Hennessy, John.** *Computer Organization and Design.* s.l. : Morgan Kauffman, 2007.
18. Tutorial: Integrar SESC a la IDE KDevelop 3. [En línea] <http://ce.sharif.edu/~haghdoost/mypages/kdevelop.htm>.
19. **Veenstra, Jack E. y Fowler, Robert J.** *MINT: A Front End for Efficient Simulation of Shared-Memory Multiprocessor.*
20. **Andrews, G. R.** *Concurrent Programming: Principles and Practice.* Redwood City - California : Benjamin/Cummings, 1991.
21. **Stalling, William.** *Sistemas Operativos.* s.l. : Prentice Hall, 2006.
22. **Granda, Mercedes.** REDES DE PETRI: MODELADO DE SISTEMAS CONCURRENTES. Cantabria, España : Programación Concurrente Departamento de Electrónica y Computadores, Universidad de Cantabria, 2010.