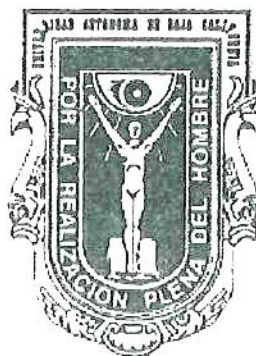


UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE CIENCIAS



**AUTOMATIZACIÓN DE LA EVALUACIÓN DE DESEMPEÑO DE
SISTEMAS TIPO SSI**

TESIS

Que para obtener el título de

LICENCIADO EN CIENCIAS COMPUTACIONALES

PRESENTA:

MASARU NATSU CÁRDENAS

Ensenada, Baja California. Febrero de 2003

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

Facultad de Ciencias

AUTOMATIZACIÓN DE LA EVALUACIÓN DE DESEMPEÑO DE SISTEMAS TIPO SSI.

Tesis Profesional

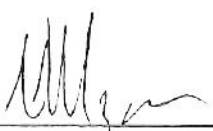
Que presenta

Masaru Natsu Cárdenas

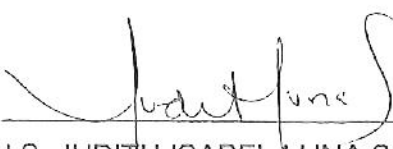
Aprobado por:


M.C. ADAN HIRALES CARBAJAL
Presidente


DR. OSCAR LÓPEZ BONILLA
Secretario


M.C. MARIA VICTORIA MEZA KUBO
Sinodal


FIS. FRANCISCO JUÁREZ GARCÍA
Suplente


M.I.S. JUDITH ISABEL LUNA SERRANO
Suplente

Resumen
de la tesis de
Masaru Natsu Cárdenas
Presentado como requisito parcial
para la obtención del título de
Licenciado en Ciencias Computacionales

Ensenada, Baja California, México.

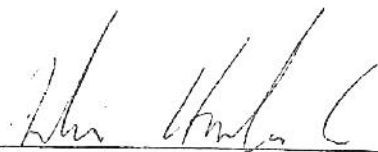
Febrero de 2003

**AUTOMATIZACIÓN DE LA EVALUACIÓN DE DESEMPEÑO DE
SISTEMAS TIPO SSI**

Resumen aprobado:

El presente trabajo presenta un modelo para la automatización del proceso de evaluación de desempeño en base al paradigma de evaluación por medición aplicada a sistemas de imagen única.

Esto sustentó el desarrollo de una herramienta que facilita la adquisición de conteos de instrucciones generadas por microprocesadores en una arquitectura distribuida. Tal sistema, requiere de una biblioteca *wrapper* generada como producto de este trabajo, la cual automatiza la recopilación de los conteos de instrucciones. En el desarrollo de esta investigación, se utiliza el sistema operativo Linux, MPI como biblioteca de comunicación y un cluster de 4 nodos Intel Dual-PIII como sistema de imagen única.



M.C. ADAN HIRALES CARBAJAL
Director del Trabajo

Agradecimientos

A mis padres, Jesús Alejandro Natsu Valencia y Macarena Cárdenas Rodrigo, que son las personas que han hecho de mi el hombre que ahora soy, por todo su apoyo y por creer en mi todo el tiempo, muchas gracias.

A mi hermano, Hiroshi Natsu Cárdenas, por mantener un espíritu de competencia entre nosotros que nos ha llevado a superarnos más día tras día, muchas gracias.

A mi novia y compañera, Blanca del Carmen Román Nava, por darme su amor y consejos, su compañía y muchas alegrías en todo momento, muchas gracias.

A mi asesor y amigo, Adan Hiraes Carbajal, por enseñarme lo que una tesis significa y por permitirme aprender junto con el en este trayecto, así como darme la oportunidad de desarrollar este trabajo juntos, muchas gracias.

A mi maestro y amigo, Francisco Juárez García, por ser uno de los mejores amigos que he conocido, y por todo su apoyo, muchas gracias.

A mi maestro y amigo, Jesús Ramón Lerma Aragón, por sus sabios consejos, confianza y amistad durante todos estos años.

A mi amigo, Ernesto Campos González, por su apoyo, consejos y ayuda en los momentos más difíciles, muchas gracias.

A mi maestra y amiga, Judith Isabel Luna Serrano, por su dedicación, enseñanzas y paciencia, muchas gracias.

A mis sinodales, Oscar López Bonilla y Victoria Meza Kubo, por su dedicación en este trabajo, así como el tiempo dedicado, muchas gracias.

Dedicatoria

Le dedico este trabajo a mi padre Alejandro, por que sin su apoyo y consejos este trabajo y mi carrera nunca se hubieran terminado, soy lo que soy gracias a ti. Muchas gracias Papá.

Parte de este trabajo es de mi madre Macarena, que sin su amor y paciencia jamás se hubiera terminado esta etapa de mi vida. Muchas gracias Mamá.

Le dedico también este trabajo a mi hermano Hiroshi, por ser mi hermano y fuente de inspiración para seguir superándome, te quiero.

También quiero dedicar este trabajo a mi único Amor Blanca por toda su comprensión, amor, compañía y por hacer mis días mejores a cada momento. Te amo.

En especial quiero dedicar este trabajo a dos personas que aunque no se encuentran conmigo siguen siendo fuente de inspiración en cada decisión que hago, por la manera en que cambiaron mi vida y la de muchos, así como todos los conocimientos, bromas, actividades y momentos vividos que jamás serán olvidados. Gabriel Delgado Carbellido y Marcos Guzmán descansen en Paz.

Contenido

I Introducción	1
I.1 Antecedentes	3
I.2 Justificación	8
I.3 Objetivo general	9
I.3.1 Objetivos específicos	9
I.4 Marco teórico	9
I.5 Metodología	10
I.6 Delimitación	13
II Análisis y Diseño	16
II.1 Descripción del Problema	16
II.2 Modelo de Casos de Uso	18
II.2.1 Caso de uso Contexto del Experimento	18
II.2.2 Caso de uso Definir Topología	19
II.2.3 Caso de Uso Alta	20
II.2.4 Caso de Uso Ejecutar y Verificar	22
II.2.5 Caso de uso Baja	23
II.3 Modelo Estático	23
II.4 Estructura de Objetos	24
II.5 Modelo Dinámico	26
II.5.1 Diagrama de Colaboración, Contexto del Experimento	26
II.5.2 Diagrama de Colaboración, Topología Lógica	29
II.5.3 Diagrama de Colaboración, Alta	29
II.5.4 Diagrama de Colaboración, Ejecutar y Verificar	33
II.6 Consolidación de los Diagramas de Colaboración	35
II.7 Estructura de Subsistemas	35
II.8 Diseño	45
III Implementación	47
III.1 Descripción	48
III.2 Esquemas de la implementación del código	50

III.2.1 No distribuido	50
III.2.2 Distribuido	55
III.3 Escribiendo definiciones de sobrecubierta	57
III.4 Generando la biblioteca	58
III.5 Interfase	60
III.5.1 Pantalla principal	60
III.5.2 Experimentos distribuidos	63
III.5.3 Experimentos no distribuidos	68
IV Análisis Experimental	71
IV.1 Descripción de las cargas de trabajo	72
IV.1.1 Pi	73
IV.1.2 Mandelbrot	74
IV.1.3 Matriz Transpuesta	75
IV.1.4 Integral $\cos(x)$	77
IV.1.5 Laplace	79
IV.2 Ejecución de los experimentos	80
IV.3 Resultados	81
V Conclusiones	86
V.1 Trabajo futuro	87
Anexo A , Manual de usuario	88
Anexo B , Wapper. Generación automática de bibliotecas perfiladas	99
Anexo C , PCL. Plataforma Intel Pentium Pro, P II, P III y P IV	109
Anexo D , Código de PEVTO	126
Bibliografía	156

Figuras

Figura 2.1: Casos de uso para el modulo ADE	20
Figura 2.2: Caso de uso para el modelo ACS	21
Figura 2.3: Caso de uso para el modelo AEE	22
Figura 2.4: Caso de uso para la baja	23
Figura 2.5: Modelo Conceptual Estático del Dominio del Problema	25
Figura 2.6: Diagrama de objetos del contexto de <i>PEVTO</i>	27
Figura 2.7: Diagrama de colaboración, Caso de uso Contexto del Experimento ..	28
Figura 2.8: Diagrama de colaboración, Caso de uso Topología Lógica	30
Figura 2.9: Diagrama de colaboración, Caso de uso Alta	32
Figura 2.10: Diagrama de Consolidación, Caso de uso Ejecutar y Verificar	36
Figura 2.11: Diagrama de Consolidación de los Diagramas de Colaboración	37
Figura 2.12: Diagrama de Colaboración de Subsistemas	39
Figura 2.13: Diagrama de Subsistema Puerto	40
Figura 2.14: Diagrama de Subsistema Usuario	42
Figura 2.15: Diagrama de los Subsistemas MPI y Coordinador	43
Figura 2.16: Diagrama de Subsistema Despliegue	43
Figura 2.17: Diagrama de Subsistema Nodos	44
Figura 2.18: Subsistema original <i>MPI</i>	45
Figura 2.19: Subsistema modificado <i>PMPI</i>	45
Figura 3.1: Pantalla principal de la interfase de <i>PEVTO</i>	61
Figura 3.2: Menú principal	62
Figura 3.3: Menú de ayuda	62
Figura 3.4: Selección de carga distribuida	64
Figura 3.5: Ventana de selección de archivos	66
Figura 3.6: Ventana de contadores	67
Figura 3.7: Ventana de ruta de bibliotecas	68
Figura 3.8: Selección de cargas no distribuidas	69
Figura 3.9: Selección y generación de cargas no distribuidas	70
Figura 3.10: Ruta de bibliotecas <i>PCL</i>	70
Figura 4.1: Representación de <i>Pi</i>	73

Figura 4.2: Partición en 1-D de una matriz de 4x4	76
Figura 4.3: División de j matrices en i procesadores	77
Figura 4.4: Integral del $\cos(x)$	78
Figura 4.5: Vector de desempeño utilizando LOADSTORE_INSTR, L2DCACHE_WRITE, JUMP_SUCCES	82
Figura 4.6: Vector de desempeño utilizando L2CACHE_READWRITE, ELAPSED_CYCLES y JUMP	83
Figura 4.7: Vector de desempeño utilizando FP_INSTR, L2CACHE_READ y L2ICACHE_READ	84
Figura 4.8: Vector de desempeño utilizando L2CACHE_READWRITE, ELAPSED_CYCLES y JUMP	85
Figura 'Ventana Principal': Figura de la ventana principal de <i>PEVTO</i>	88
Figura 'Menú Principal': Figura del menú principal de la aplicación	89
Figura 'Menú Ayuda': Menú de ayuda de la ventana principal	90
Figura 'Cargas distribuidas': Ventana con los programas paralelos disponibles .	91
Figura 'Selección de cargas distribuidas': Ventana de selección de cargas	92
Figura 'Selección de contadores': Ventana que muestra los posibles contadores	93
Figura 'Ruta de bibliotecas paralelas': Ventana de rutas para la fase de enlace	94
Figura 'Cargas no distribuidas': Ventana con los programas no paralelos disponibles	95
Figura 'Selección de cargas no distribuidas': Ventana de selección de cargas	97
Figura 'Rutas de bibliotecas no paralelas': ventana de rutas para la fase de enlace	98

Tablas

Tabla 1: Eventos que pueden ser medidos en PCL	121
--	-----

I Introducción

En la actualidad existen muchas maneras de medir el desempeño de los sistemas distribuidos, algunas son por medio del tiempo. Es decir, que se toman los tiempos de ejecución y se busca llegar a un tiempo que se considera óptimo, otros buscan aplicar cargas y analizar como se desempeñan en el sistema, los sistemas más comunes que se buscan analizar son las super computadoras y los sistemas tipo cluster. Las super computadoras son máquinas que contienen dos o más procesadores, grandes cantidades de memoria y mucha capacidad de almacenamiento capaces de procesar grandes cargas de trabajo y los sistemas tipo cluster son máquinas conectada entre sí que usan la comunicación por paso de mensajes y que el usuario las percibe como un solo sistema, también conocidos como sistemas de imagen única.

La presente investigación implementa una herramienta para la evaluación de desempeño de sistemas de imagen única. Partiendo del proceso que actualmente se realiza para la evaluación de desempeño y con base en los antecedentes del mismo se define el problema y los objetivos a lograr, los cuales han servido para diseñar la metodología del presente trabajo.

A través de un análisis de requerimientos y especificaciones se muestra una

problemática a tratarse de una manera gráfica, haciendo énfasis en los diversos aspectos del sistema como son: Comunicación, modularidad, interrelación con el usuario, etc.

Se describe de manera técnica y detallada los pasos lógicos necesarios para cumplir los requerimientos planteados en el análisis del sistema. Tomando en cuenta los objetivos y la metodología implementada se diseñan los experimentos que describen las características y comportamiento del problema, siendo esta fase fundamental para la obtención de información, los cuales se analizan mediante un método analítico, proporcionando información descriptiva de la arquitectura y su operabilidad.

Basándose en los procedimientos actuales de análisis de desempeño y debido a los costos y tiempos de los procesos de evaluación, se sugiere el uso y aplicación de esta herramienta como una opción que facilite el análisis de desempeño.

Uno de los retos más importantes en la evaluación de desempeño de sistemas tipo cluster, es su automatización, dicho objetivo no es conveniente llevarlo a cabo utilizando los procedimientos actuales que proporcionan una descripción de alto nivel. Para el análisis de sistemas distribuidos de imagen única es necesario obtener información más completa y detallada.

El uso de esta herramienta facilita la caracterización del comportamiento de las distintas arquitecturas distribuidas en especial clusters de imagen única. Asimismo despertar el interés por el uso y aplicación de la misma.

I.1 Antecedentes.

Los antecedentes bibliográficos que a continuación se presentan están relacionados con las herramientas y conceptos que apoyan el presente trabajo de investigación.

En la actualidad cada vez es más frecuente el término "evaluación de desempeño", es incluso un método de control en el desarrollo de *software* y *hardware*. La mayoría de los usuarios de arquitecturas tipo *cluster*¹ encuentran la evaluación de rendimiento difícil de elaborar debido a lo que envuelve el realizar dicha actividad.

Definiremos el término desempeño como: la ejecución de una acción; el cumplimiento satisfactorio o no satisfactorio de una actividad o petición. Contemplando como principal objetivo la ganancia de beneficio en comparación con un estado previo. Por ejemplo, disminución en tiempo de ejecución. Algunas de las razones por las que esta evaluación puede ser complicada son: no

¹ Un cluster puede ser definido como "una red con elementos de procesamiento, donde cada unidad de procesamiento tiene como característica principal: uno o más *CPUs* y memoria local. Las unidades de procesamiento no comparten memoria y se comunican a través de un protocolo de paso de mensajes" [Rajkumar Buyya, 1999].

contar con herramientas que ofrezcan información sobre el comportamiento interno de la arquitectura, las actuales herramientas solo se limitan a un parámetro de evaluación por lo que son específicas, la evaluación de desempeño es un procedimiento costoso en cuanto al tiempo que el usuario debe invertir y el conocimiento que este requiere.

Dos técnicas utilizadas para la evaluación de desempeño son: evaluación por modelado y evaluación por medición. Jain Anil considera al modelado analítico como una tercer clasificación [Jain Anil, 1991]. La técnica de evaluación por modelado permite simular el desempeño mediante la utilización de un modelo del sistema en vez de la arquitectura o aplicación a evaluar. Esta técnica es compleja de implementar y la simulación, dependiendo en el detalle de la implementación, puede ser costosa de ejecutar. Nuevas estrategias para la simulación consisten en la implementación de una máquina virtual² del sistema en estudio [Balderas Contreras y García Monroy, 2000].

La técnica de evaluación por medición consiste básicamente en una secuencia de dos pasos: recabar información de desempeño del sistema durante la ejecución de su carga de trabajo, y posteriormente, utilizar algún paradigma de la evaluación de desempeño para procesar la información recabada. Esta técnica también se consideraba como una estrategia costosa

² Una máquina virtual es un programa que contiene toda la funcionalidad de un sistema, pero se puede ejecutar sobre distintas arquitecturas.

debido que el perfilar la ejecución de una aplicación era normalmente realizada a nivel del núcleo mediante el uso de interrupciones al sistema operativo. Esto tiene severas implicaciones, teniendo como resultado mediciones con menor precisión, debido a que muchas operaciones no son consideradas durante los periodos de interrupción. La evaluación por medición ha venido siendo una estrategia más factible, debido a la implementación de nuevas técnicas para realizar el perfilado. Por ejemplo, actualmente se implementan a nivel microprocesador de PC's, estaciones de trabajo y de super computadoras contadores de instrucciones [Berrendorf Rudolf y Ziegler Heinz, 2000].

En este trabajo nos enfocamos en el área de evaluación de desempeño de sistemas distribuidos, en particular de sistemas con una imagen única (*Single System Image, SSI*) [Rajkumar Buyya, 1999], con orientación a la evaluación de desempeño por medición.

Quizás el paradigma más fundamental y de mayor controversia para la evaluación de desempeño de sistemas paralelos es la ley de Amdahl [Allen Malony, 2000]. Su propósito principal es señalar el incremento de desempeño (*speedup*) de una aplicación tomando en cuenta únicamente dos parámetros: la fracción de la aplicación que se ejecuta de manera secuencial y el número de procesadores utilizados para procesar la fracción paralela de la aplicación. Otras especializaciones para la Ley de Amdahl han sido desarrolladas, por

ejemplo: la ley de Gustafson y la ley de Sun y Ni [Hwang Kai y Zhiwei Xu, 1998]. Sin embargo, el incremento de desempeño es un paradigma pobre para la investigación de problemas de rendimiento.

Los *benchmarks*³ han sido aceptados ampliamente como una técnica para la evaluación de desempeño debido a esfuerzos que se han invertido para hacerlos portables, de amplia disposición, delimitan cargas predefinidas de trabajo y por su facilidad de uso. Los *benchmarks* han sido utilizados para más que solo estimar el desempeño promedio de sistemas, su principal aplicación es el proporcionar una estrategia independiente de cualquier arquitectura para comparar sistemas candidatos.

Un *benchmark* calcula el desempeño de un sistema basándose en el conteo de operaciones y presenta al usuario un promedio de la operación que se a monitoreado, por ejemplo, *FLOPS*, promedio de operaciones *TLB*. Sin embargo, tal promedio es una abstracción de alto nivel que describe el desempeño promedio del sistema y es un parámetro pobre para propósitos específicos del análisis de desempeño.

Allen Malony en [Allen Malony, 2000] sugiere que un análisis de desempeño con mayor precisión es asistido por "Un ambiente para solucionar problemas de

³ Un benchmark es un aplicación que evalúa la capacidad de procesamiento y administración de un sistema o subsistema bajo una carga bien definida de trabajo.

desempeño, compuesto por: una metodología paralela aplicada para la evaluación de rendimiento y de un conjunto integrado de herramientas para el modelado, medición, análisis, presentación y predicción de desempeño". Tal ambiente puede hacer uso de metodologías híbridas de manera cíclica o no cíclica para generar un análisis detallado de datos que describan el desempeño de un sistema.

Debido a deficiencias presentes en paradigmas de evaluación de desempeño y a la amplia aceptación de herramientas tales como los *benchmarks*. Scherson [Scherson Isaac y Krishnaswamy Umesh, 1993] [Krishnaswamy Umesh, 1995] [Scherson Isaac y Krishnaswamy Umesh, 1996] propone un modelo para la evaluación de desempeño de sistemas de memoria compartida denominado el modelo geométrico de ejecución de máquina (*Geometric Model of machine execution, GM*). El modelo geométrico de ejecución permite modelar el desempeño de un sistema a una granularidad fina basándose en tres componentes: puntos de soporte, puntos operacionales y un punto medio operacional. Los puntos de soporte representan todos los posibles tiempos de ejecución de una instrucción. Un punto operacional representa el promedio de los puntos de soporte de una instrucción, por lo tanto, los puntos operacionales representan el promedio de ejecución de un conjunto de instrucciones. El punto medio operacional es el punto promedio de los puntos de los puntos operacionales y en conclusión describe el desempeño promedio del sistema. Tal

punto se representa como un vector de mediciones, por ejemplo, promedio de lecturas al *TLB*, promedio de hits en memoria *L1* o *misses*, etc. Se propone el método de los puntos de intersección para poder estimar el vector de mediciones (*Intersection Point Method, IPM*).

El *IPM* también sirve como herramienta para la caracterización de la carga de trabajo, asociando a un *benchmark* una medida cuantitativa de calidad denominada aptitud del *benchmark*. Sin embargo, el procedimiento para estimar el vector de rendimiento presenta una limitación cuando la información es linealmente dependiente. En el presente trabajo, y tomando en cuenta estas características se llega al criterio de adoptar al *MOP* (Punto Medio Operacional) y al *IPM* como métodos para obtener una interpretación de los resultados de los experimentos.

1.2 Justificación.

La evaluación de desempeño es una actividad de mucha importancia dentro del computo distribuido, actualmente esto implica un proceso manual, monótono e incluso repetitivo, con un alto costo de conocimiento necesario para que pueda ser realizado. Esto presenta una gran desventaja en sistemas de gran escala, ya que la evaluación involucraría demasiado tiempo para su realización, debido a que debe llevarse a cabo de manera individual en cada uno de los nodos que conforman el sistema.

Lo anterior se considera un verdadero problema a solucionar. El principal objetivo de la presente investigación es proporcionar una herramienta que automatice el proceso de evaluación de desempeño y que facilite la adquisición de la información.

I.3 Objetivo general.

Construir una herramienta que permita evaluar el desempeño de un sistema tipo "*Single System Image*" (*SSI*), en particular un cluster.

I.3.1 Objetivos específicos.

- Que la granularidad de la evaluación sea a nivel nodo y proceso.
- Generar una descripción vectorial de acuerdo al nivel de granularidad especificado.
- Que sea aplicable a distintas arquitecturas.

I.4 Marco teórico.

Sistema Distribuido: Según Andrew S. Tanenbaum [Sistemas operativos distribuidos, 1996], "*Un sistema distribuido es una colección de computadoras independientes que aparecen ante el usuario del sistema como una única computadora*".

Cluster: Según el *Scalable Computing Laboratory*, "*Un cluster es un grupo de sistemas de computadoras independientes y estas forman un sistema multiprocesadores ligeramente acoplado. La red es usada para proveer comunicación entre procesadores*", [<http://www.scl.ameslab.gov/Projects/ClusterCookbook/>].

Según Neil Matthew y Richard Stones [Professional Linux Programming, 2001], "*Un cluster consiste en un grupo de computadoras interconectadas por una red para formar un sistema fuertemente acoplado o un sistema computacional de memoria compartida*".

Modelo de programación por paso de mensajes: Según Neil Matthew y Richard Stones [Professional Linux Programming, 2001], "*Consiste en un grupo de procesos, cada uno trabajando en un subconjunto de datos. Los procesos se comunican unos con otros usando mensajes para acceder y modificar datos que pertenecen a otros procesos*".

Benchmark: Es un aplicación que evalúa la capacidad de procesamiento y administración de información de un sistema o subsistema bajo una carga bien definida de trabajo.

I.5 Metodología.

Con el propósito de lograr los objetivos de la presente investigación se

diseñaron los diferentes aspectos metodológicos.

Se requirió realizar la Instalación y configuración de un ambiente tipo *cluster*. Para ello se tuvo que establecer conexión física entre 3 o 4 computadoras, Instalar el sistema operativo *Linux* en las máquinas e Instalar el *software* necesario para el paso de mensajes, seleccionamos *MPI (MPICH)* puesto que es un estándar en el paso de mensajes.

Preparación del ambiente de trabajo. Lo cual llevó a la Instalación de las bibliotecas que permiten el acceso a los registros del microprocesador, en particular *PCL, Performance Counter Library*, agregar líneas al código del núcleo del sistema operativo (*kernel*) para considerar la lectura de contadores. Esto se hará mediante el uso de parches al *kernel*, recompilación del *kernel* para obtener una imagen del mismo con la lectura de registros activada y disponible.

Instrumentación de código. Con el objeto de desarrollar archivos de sobre cubierta para generar código fuente con capacidad para leer los contadores del microprocesador, se crea un grupo de bibliotecas nuevas a partir del código instrumentado, de manera que abarquen todos los registros de desempeño disponibles dentro de la arquitectura.

Selección de cargas de trabajo. Para esto se selecciona un conjunto de

programas paralelizados con directivas de *MPI*, estos se recompilan con las bibliotecas instrumentadas, tal que se obtiene un código ejecutable instrumentado. Las carga de trabajo seleccionadas solucionan distintos problemas por lo que en principio se consideran heterogéneas.

Implementación de la interfase de usuario. En la implementación de la interfase de usuario se selecciona un lenguaje de implementación con un esquema orientado a objetos y que permita enlazarse con las bibliotecas ya generadas, y agregar un sistema de selección de cargas disponibles para el usuario.

Desarrollo del sistema tolerante a fallas. En esta etapa de la metodología se necesita diseñar un servidor de nodos disponibles así como elaborar un módulo que sirva como coordinador donde su principal función consiste en: Tener la habilidad de decisión en caso de que un nodo falle.

Otras funciones que realizan los participantes dentro de un escenario tolerante a fallas son: Reporte de fallo por parte del coordinador y la elección de un nuevo coordinador en caso de que este falle.

Se recomienda que el subsistema que de solución a estos requerimientos sea implementado en *CORBA*, debido a que este es un estándar de

comunicación sin importar la heterogeneidad del *cluster*.

Implementación de un método de caracterización de resultados. Se realizó la implementación del *MOP* (*Mead point operation*), como método para la interpretación de resultados recabados durante la evaluación, así como el uso del *Intersection point method* (*IP*), para estimar los vectores de desempeño.

Tomando en cuenta los anteriores criterios se realiza un análisis y diseño que determina la estructura del sistema.

1.6 Delimitación.

Tomando en cuenta el objetivo y las actividades propuestas así como el tiempo para las mismas, el alcance de la tesis se ha delimitado a las siguientes actividades.

1. Instalación y configuración de un ambiente tipo *cluster*.

1.1 Establecer conexión física entre 3 o 4 computadoras.

1.2 Instalación del sistema operativo *Linux* en las máquinas

1.3 Instalación del *software* necesario para el paso de mensajes, seleccionamos *MPI* (*MPICH*) puesto a que es un estándar en el paso de mensajes.

2. Preparación del ambiente de trabajo.

2.1 Instalación de las bibliotecas que permiten el acceso a los registros del microprocesador, en particular *PCL*, *Performance Counter Library*.

2.2 Agregar líneas al código del núcleo del sistema operativo (*kernel*), para considerar la lectura de contadores. Esto se hará mediante el uso de parches al *kernel*.

2.3 Re compilación del *kernel* para obtener una imagen del kernel con la lectura de registros activada y disponible.

3. Instrumentación de código.

3.1 Desarrollo de archivos de sobre cubierta para generar código instrumentado.

3.2 Genera un grupo de bibliotecas nuevas a partir del código instrumentado anteriormente, de manera que cubran todo los contadores disponibles dentro de la arquitectura.

4. Selección de cargas de trabajo.

4.1 Obtener nuevas cargas de trabajo al compilar las cargas ya seleccionadas con las bibliotecas generadas.

5. Implementación de la interfase de usuario.

5.1 Selección del lenguaje para interfase, tomando en cuenta un esquema orientado a objetos.

5.2 Enlazar la interfase con las bibliotecas ya generadas.

5.3 Generar un sistema de selección de cargas para que sean incluidas dentro del sistema disponibles para cuando las requiera el usuario.

6. Análisis, diseño e implementación parcial del sistema.

II Análisis y Diseño

Introducción.

En este caso de estudio se presenta la descripción del sistema evaluador de desempeño. En primera instancia se consideran las entradas y salidas debido a que se necesita interactuar con otros sistemas y subsistemas. El sistema recibe diferentes peticiones del usuario, y tiene que solucionar la comunicación con puertos o nodos en el entorno de red. Así como verificar la integridad del experimento y reportar los resultados.

Primero se realiza un modelo del análisis, entonces el mismo análisis es proyectado, en primer caso, a un diseño no distribuido, y en segundo caso a un diseño distribuido.

Los casos de estudio de *PEVTO*⁴ fueron desarrollados bajo los estándares del modelado UML [HA, 2000], incluyendo [Jackson, 1983], [Saden, 1994], y [Douglass, 1996b].

II.1 Descripción del Problema.

En el problema tenemos:

- Una interfase de usuario.

⁴ PEVTO: Performance Evaluation Tool, es lo que llamamos Herramienta para la evaluación de desempeño.

- Un conjunto de parámetros que definen el experimento.
- Un nodo coordinador.
- Un puerto de visualización.
- Un servidor de grupos.
- Algoritmos y programas de comunicación.

Para cada nodo en el experimento se tiene:

- Una interfase de comunicación.
- Un control propio del nodo.
- Una administración local de tabla de grupos.

Para cada nodo existe una interfase propia de comunicación que le permite establecer comunicación con el nodo o puerto coordinador así como intercambiar mensajes con el nodo o puerto central y con el *middleware* (MPI). Sólo hay un coordinador (a la vez), un solo servidor de grupos y un solo puerto de visualización.

La función principal de *PEVTO* consiste en proporcionar un ambiente para la evaluación de desempeño de arquitecturas distribuidas homogéneas y heterogéneas.

El ambiente integral de PEVTO está compuesto por tres módulos: el *Ambiente de Definición del Experimento (ADE)*, el *Ambiente de Configuración del Sistema (ACS)*, y el *Ambiente de Ejecución del Experimento (AEE)*. El ADE proporciona los medios que permitan al usuario definir la carga de trabajo, tipo y cantidad de parámetros, nivel de instrumentación, técnica de caracterización de carga de trabajo, granularidad del análisis y estrategia de visualización. El ACS debe de proveer soporte para: la definición lógica de la arquitectura donde se ejecutará la aplicación, detección automática de nodos o puertos y la actualización del mapa lógico. El AEE provee la capacidad de realizar la ejecución del experimento con los datos provistos por el ADE y el ACS, así como la verificación de la integridad de la ejecución del experimento una vez que este se encuentre en ejecución.

II.2 Modelo de Casos de Uso.

II.2.1 Caso de uso Contexto del Experimento.

Nombre: Contexto del Experimento.

Resumen: El usuario define el contexto para la ejecución del experimento.

Actor(es): Usuario, como actor primario.

Precondiciones: El sistema despliega una pantalla de selección para el ADE.

Descripción:

1. El usuario define los parámetros, nivel de instrumentación, técnica

- de caracterización, etc. Sin seguir ningún orden específico.
2. El usuario confirma que terminó con la selección.
 3. El sistema pasa a definir la topología lógica.

II.2.2 Caso de uso Definir Topología.

Nombre: Definir Topología.

Resumen: El usuario selecciona los nodos o puertos que realizarán el experimento.

Precondiciones: El sistema presenta al usuario un mapa lógico, todos los nodos en el mapa deben de ser accesibles.

Actor(es): Usuario.

Descripción:

2. El usuario selecciona los nodos o puertos que desea que participen en el experimento.
3. El sistema presenta un mensaje de confirmación una vez que el usuario terminó la selección.
4. El sistema entra en un estado transitorio de espera hasta que el usuario confirma la selección.

Alternativas:

- Si el usuario no hace la selección de nodos el experimento se realiza en todos los nodos o puertos disponibles.

Estos casos de uso son parte del módulo ADE el cual se aprecia en la figura 2.1, que se muestra a continuación.

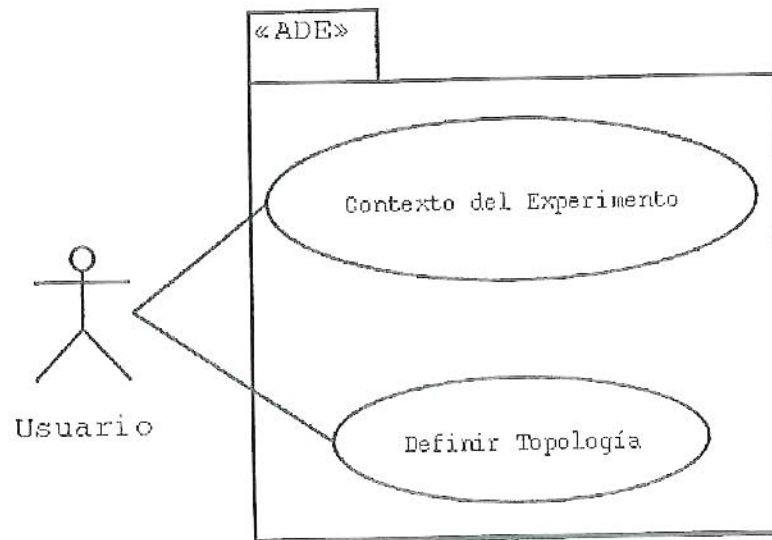


Figura 2.1: Casos de uso para el modulo ADE.

II.2.3 Caso de Uso Alta.

Nombre: Alta.

Resumen: El usuario decide que nodos o puertos formarán parte de la topología lógica del experimento.

Actor(es): Primario – Usuario, Secundario – Sistema Operativo.

Precondiciones: El software correspondiente al nodo o puerto tiene que estar previamente instalado. Y en ejecución.

Descripción:

5. Después de que la topología lógica a sido definida, se envía un

mensaje de que se desea dar de alta.

6. El nodo o puerto pide una descripción de sus propias características a su sistema operativo.
7. El sistema operativo responde con la descripción al nodo.
8. Se envía una solicitud de integración al servidor de grupos enviando su propia descripción. El nodo activa su administración local de la tabla de grupos (T.G.).
9. El servidor de grupos responde con información actual sobre la tabla de grupos, el id, y una bandera que indica si es coordinador o no.
10. Se envía a un sistema coordinador una señal de activación la cual se puede interpretar como que esta listo para participar.
11. El nodo o puerto envía los datos a la administración local, así se tiene la información actualizada.
12. La administración local actualiza la tabla de grupos local.

Este es el caso de uso para el modulo de ACS, el cual se muestra en la figura 2.2.

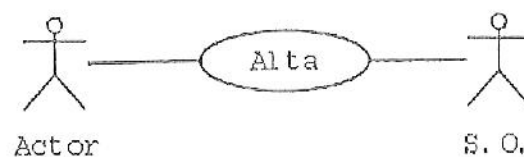


Figura 2.2: Caso de uso para el modelo ACS.

II.2.4 Caso de Uso Ejecutar y Verificar.

Nombre: Ejecutar y Verificar.

Resumen: La sesión se ejecuta y un evento periódico es generado, el evento se encarga de verificar la integridad del experimento, tomando en cuenta la disponibilidad de los nodos o puertos que integran la topología.

Actor(es): Primario – Usuario, Secundario – Sistema Operativo.

Descripción:

1. Se ejecuta la sesión del experimento.
2. Se inicia la verificación de integridad.

Alternativas:

- Si un nodo o puerto presenta algún problema, no puede responder o no puede continuar con la sesión, se reporta el problema y se termina con la sesión.

En la siguiente figura se muestra el caso de uso Ejecutar y Verificar, este es el modulo AEE.

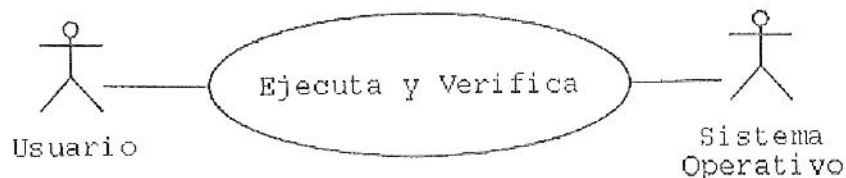


Figura 2.3: Caso de uso para el modelo AEE.

II.2.5 Caso de uso Baja.

Nombre: Baja.

Actores: Ninguno.

Precondiciones: El nodo debe darse de baja antes de iniciar la ejecución del experimento.

Descripción:

1. El nodo simplemente se excluye por cualquier método de la ejecución, el sistema se encargará de resolver su ausencia.

En la siguiente figura se muestra el caso de uso Baja.

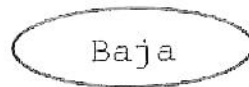


Figura 2.4: Caso de uso para la baja.

II.3 Modelo Estático.

A través del modelo estático se captura la relación entre las clases del dominio del problema, como se muestra en la figura 2.5. En este diagrama se aprecian las interrelaciones entre las diversas clases como son: **Nodos**, la cual contiene a la de **Comunicación nodos** y a la de **Control nodos**, y a su vez hereda de **coordinador**, sus características, cualidades, operaciones y funciones. Asimismo el **Puerto control**, se comunica con **Nodos** para enviar y

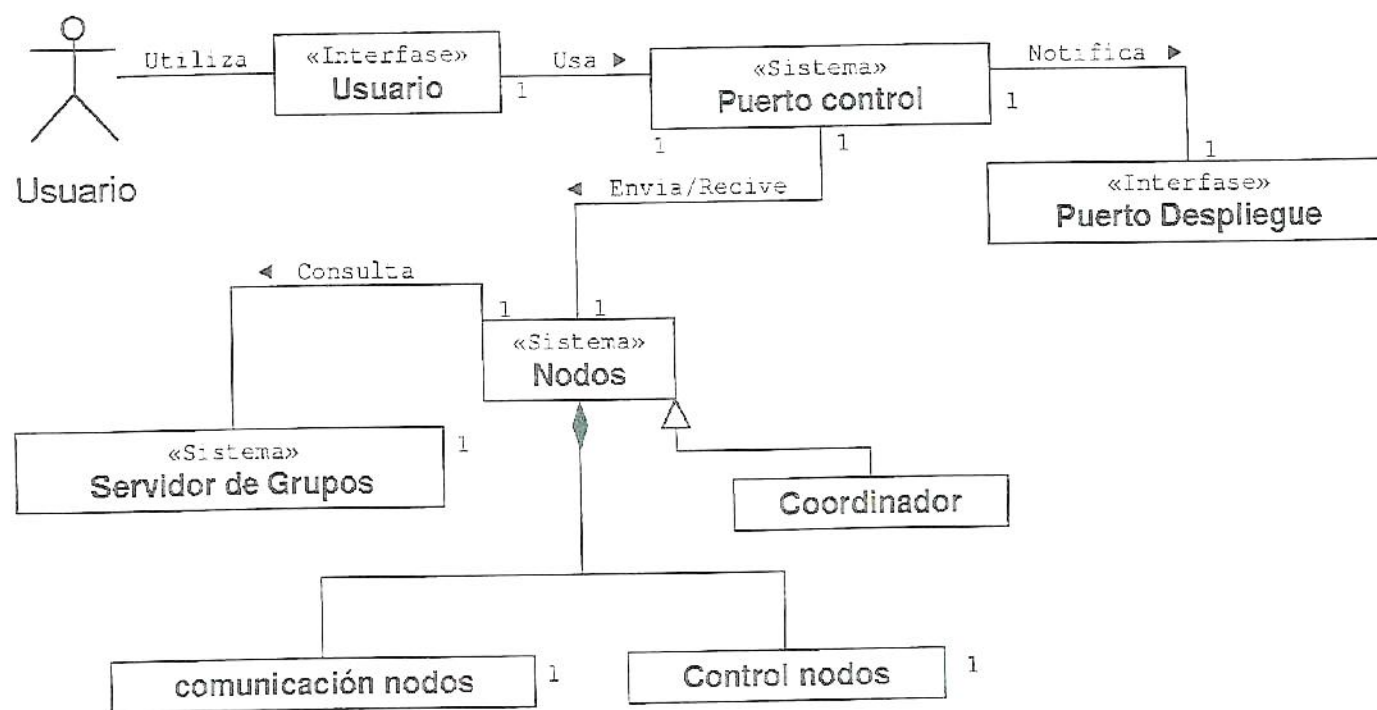
recibir información de todo tipo, tanto del experimento como de la integridad y su continuidad.

Puerto control mantiene una comunicación con ***Usuario*** para enviar y recibir datos, y a su vez dirigir los resultados a ***Puerto despliegue***.

II.4 Estructura de Objetos.

Ahora consideraremos los objetos en el caso de estudio de *PEVTO* para el desarrollo del modelo dinámico, para cada interfase existe un objeto constitutivo. Así para cada interfase externa existe un objeto externo con el que interactúa. Por ejemplo, la interfase de usuario interactúa con el usuario, la interfase de despliegue interactúa con el puerto de despliegue, la interfase de comunicación de nodos interactúa con los nodo o puertos participantes, y se puede considerar en este caso la interfase de sistema operativo.

Figura 2.5: Modelo Conceptual Estático del Dominio del Problema.



Considerando únicamente los objetos y modelando su interacción con el control del sistema, como se muestra en la figura 2.6.

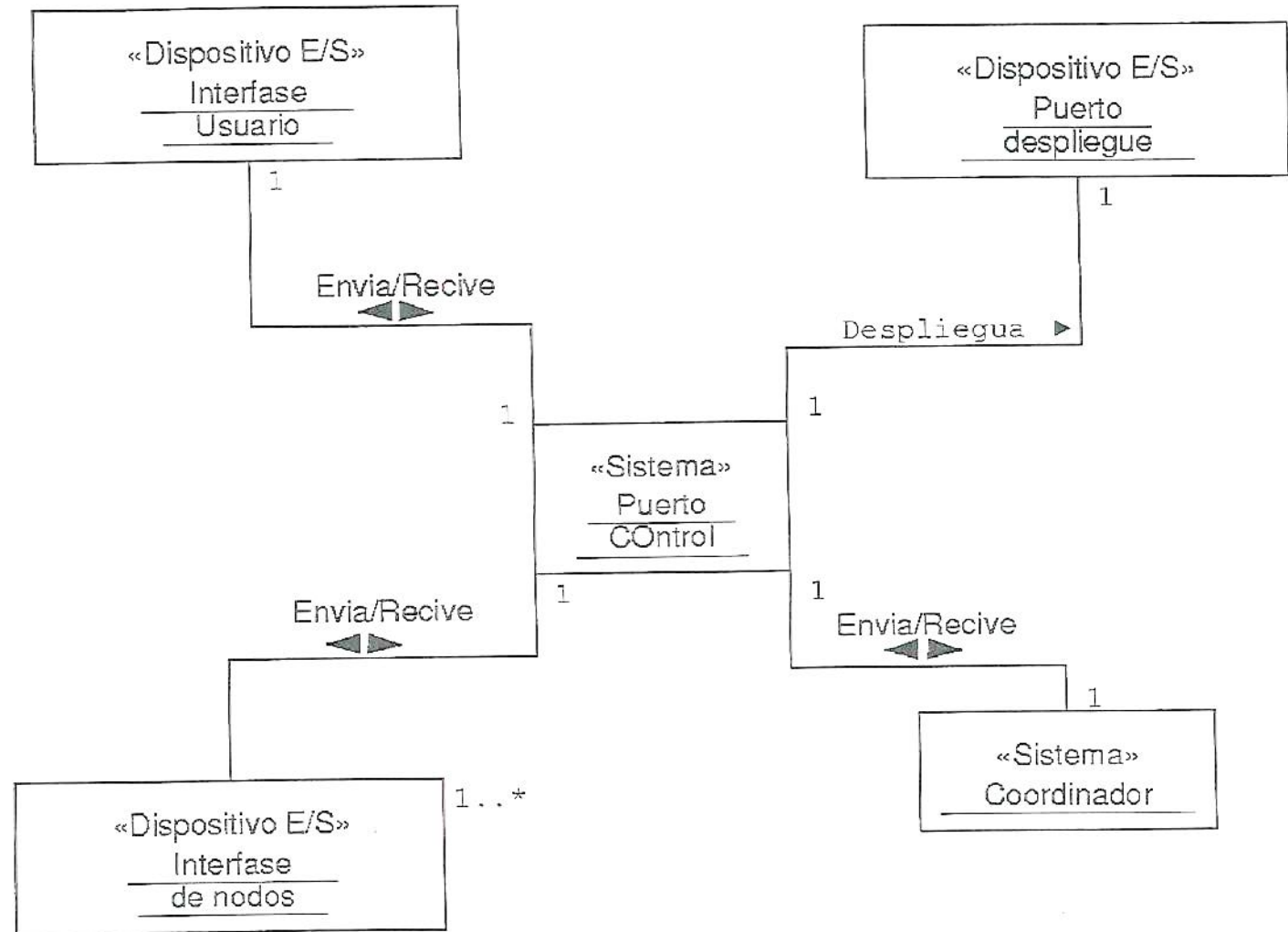
II.5 Modelo Dinámico.

En el modelo dinámico se define la interacción de los objetos para cada caso de uso. En este caso de estudio, se usan los diagramas de colaboración. Un diagrama de colaboración es desarrollado para representar los objetos que participan en cada caso de uso y su secuencia de interacciones.

II.5.1 Diagrama de Colaboración, Contexto del Experimento.

EL diagrama de colaboración para el caso de uso Contexto del Experimento está representado en la figura 2.7. El contexto del experimento no depende de ningún otro caso de uso para llevarse a cabo, es por eso que es el primer caso de uso en llevarse a cabo. Una selección aleatoria por parte de usuario es hecha con respecto al contexto del experimento entonces el sistema pasa a seleccionar la topología lógica, esto se realiza por parte del usuario no es necesaria la intervención el resto del sistema, se le presentan unas pantallas al usuario, el realiza su selección y entonces el sistema inicia un nuevo caso de uso.

Figura 2.6: Diagrama de objetos del contexto de PEVTO.



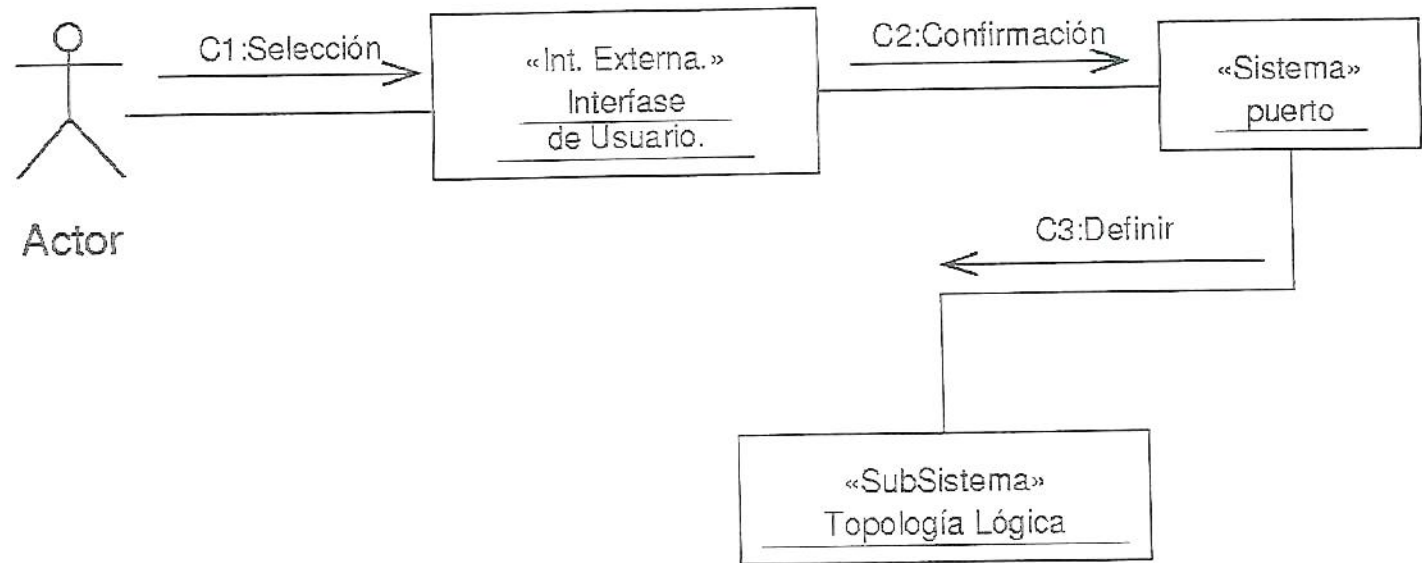


Figura 2.7: Diagrama de colaboración, Caso de uso Contexto del Experimento.

Secuencia de Mensajes:

C1: El usuario selecciona aleatoriamente los parámetros y define el contexto del Experimento.

C2: El usuario confirma su definición de parámetros.

C3: El sistema pasa a definir la topología lógica.

II.5.2 Diagrama de Colaboración, Topología Lógica.

En la figura 2.8, se encuentra el diagrama de colaboración para el caso de uso Topología Lógica. Aquí el usuario selecciona donde desea que se realice el experimento, es decir en que nodos desea que el experimento se lleve a cabo.

Secuencia de Mensajes:

T1: Se realiza la selección de nodos por el usuario.

T2: Cuando se lleva a cabo la selección los datos son pasados a un generador de lista el cual genera la lista de máquinas que participarán.

II.5.3 Diagrama de Colaboración, Alta.

El diagrama de colaboración de la Alta, en la figura 2.9, se aprecia como el usuario da de alta los diferentes nodos que previamente fueron seleccionados en la topología lógica y la comunicación que se envuelve en la alta de los nodos.

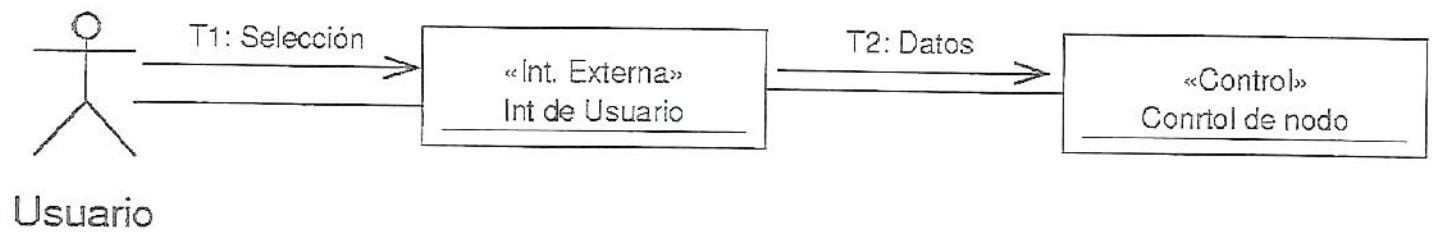


Figura 2.8: Diagrama de colaboración, Caso de uso Topología Lógica.

Secuencia de Mensajes:

A0: El usuario activa el alta en su interfase.

A1: La señal llega al nodo.

A2: El nodo consulta una descripción local a su sistema operativo a través de la interfase del sistema operativo.

A3: El nodo recibe su descripción.

A4: El nodo solicita integrarse al experimento comunicándose con el servidor de grupos y Envía su propia descripción.

A5: El servidor de grupos responde con una serie de datos como; id, t.g, etc.

A6: Si el servidor de grupos indica al nodo que es coordinador entonces se activa como tal.

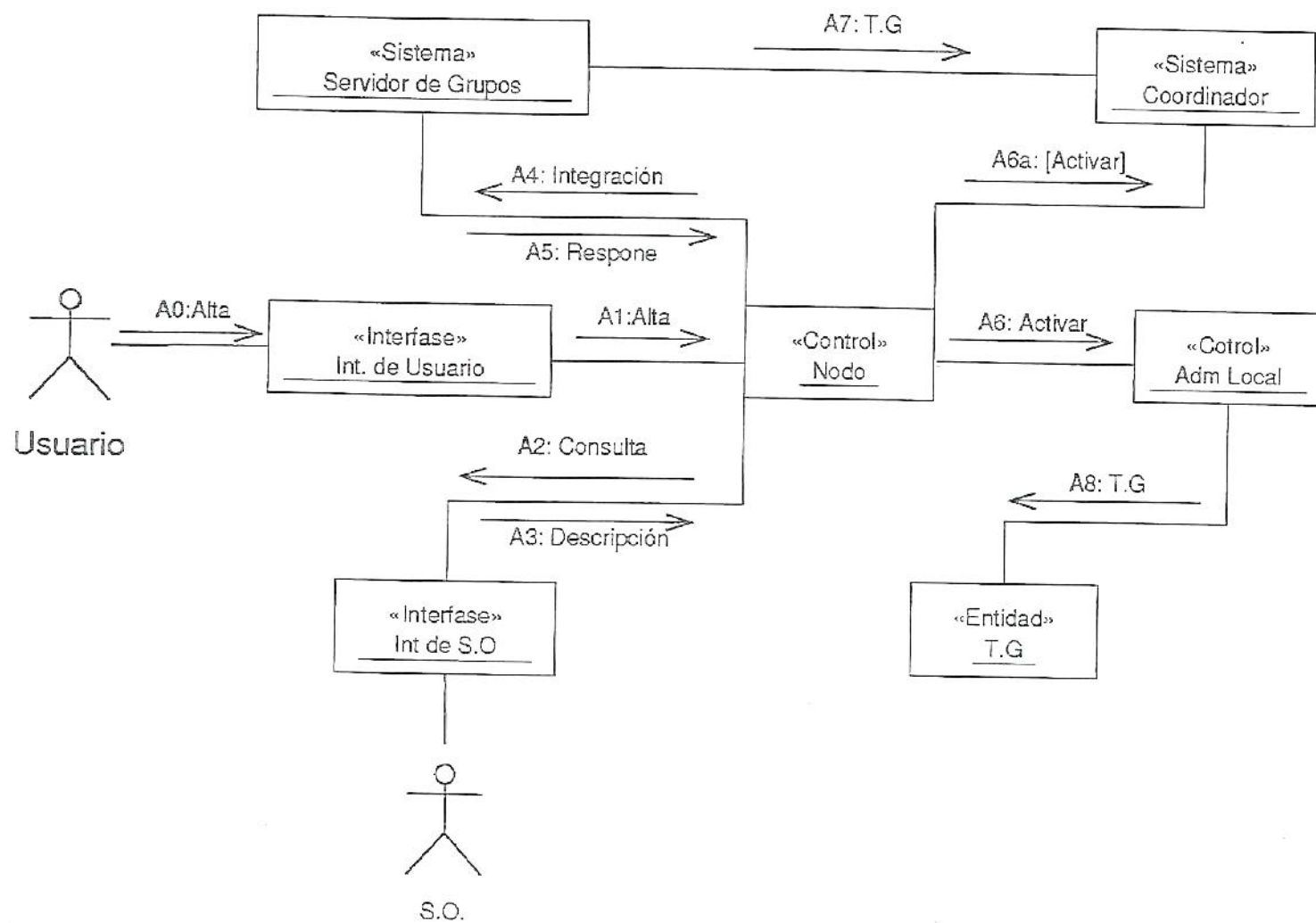
A6a: Al mismo tiempo el nodo activa su administración local.

A7: El servidor de grupos avisa a su denominado coordinador que se agrego un nuevo nodo y le pasa la nueva tabla de grupos, esto se hace para cada nodo que se da de alta.

A8: EL coordinador se encarga de mandar la nueva tabla de grupos a todos los nodos que aparecen en la tabla.

A9: La administración local de cada nodo recibe la nueva tabla y la actualiza.

Figura 2.9: Diagrama de colaboración, Caso de uso Alta.



II.5.4 Diagrama de Colaboración, Ejecutar y Verificar.

El Diagrama de colaboración Ejecutar y Verificar, es uno de los más importantes ya que muestra el proceso que se sigue cuando un experimento se está ejecutando y cuando los posibles casos de error ocurren. Este corresponde al caso de uso Ejecutar y Verificar como se muestra en la figura 2.10.

Secuencia de Mensajes:

- X1 y X2:** El usuario da la orden de ejecución a través de la interfase de usuario al control de usuario.
- X3:** El control de usuario pasa los datos del experimento.
- X4:** El puerto a través de un algoritmo solicita que se genere el archivo de las máquinas.
- X5:** El algoritmo consulta la tabla de grupos.
- X6:** El algoritmo adquiere la tabla de grupos con la que genera el archivo.
- X7:** El algoritmo genera el archivo.
- X8:** El puerto pasa los datos de ejecución y el archivo de máquinas al *MPI*.
- X9:** EL *MPI* ejecuta el experimento en todos y cada uno de los nodos del archivo.
- X10:** Los nodos terminan y reportan resultado a través del *MPI*.

- X11:** El puerto de control obtiene los resultados.
- X12:** A los resultados se les aplica un método de caracterización.
- X13:** Los resultados son reportados a la interfase de despliegue.
- X14:** Los resultados son desplegados al puerto de despliegue.
- X15:** En caso de que un nodo no responda por algún motivo el coordinador lo nota.
- X16:** El coordinador reporta que hay un problema al puerto.
- X17:** El puerto reporta el problema a la interfase de despliegue.
- X17a:** El puerto le pide al *MPI* que se detenga por causa de un problema.
- X18:** El problema es desplegado.
- X19:** En caso de que el coordinador no conteste la interfase de comunicación del nodo lo detecta.
- X20:** El nodo se da cuenta del problema.
- X21:** El nodo que detecto el problema comienza con una elección de un nuevo coordinador.
- X22:** Se envía el paquete de elección a todos los nodos de la tabla de grupos.
- X23:** El paquete regresa después de que todos aportaron sus datos.
- X24:** El nodo recibe el paquete.
- X25:** Se realiza la elección el nuevo coordinador.

X26: Se reporta nuevo coordinador y nueva tabla de grupos.

X27: Los nodos reciben quien es el nuevo coordinador y la nueva tabla de grupos.

La ejecución continua, el coordinador, mientras se encuentre activo, estará verificando la disponibilidad de los nodos.

II.6 Consolidación de los Diagramas de Colaboración.

La consolidación de los diagramas de colaboración muestran todos los objetos que participan en los diagramas de casos de uso y sus interacciones entre ellos. Algunos objetos que aparecen en mas de un caso de uso. Por ejemplo: Interfase de usuario y el control del puerto. Pero a pesar de eso sólo se muestra una vez en el diagrama consolidado, el diagrama consolidado se muestra en la figura 2.11.

II.7 Estructura de Subsistemas.

Ahora lo que se pretende es dividir el sistema en subsistemas, debido a que es una aplicación distribuida. Para determinar los subsistemas se aplicaron varios criterios, pero conservando la cualidad de que cada subsistema puede ser ejecutado en distintos nodos, UML [HA, 2000]. Aquí se comienza a proyectar a un diseño distribuido.

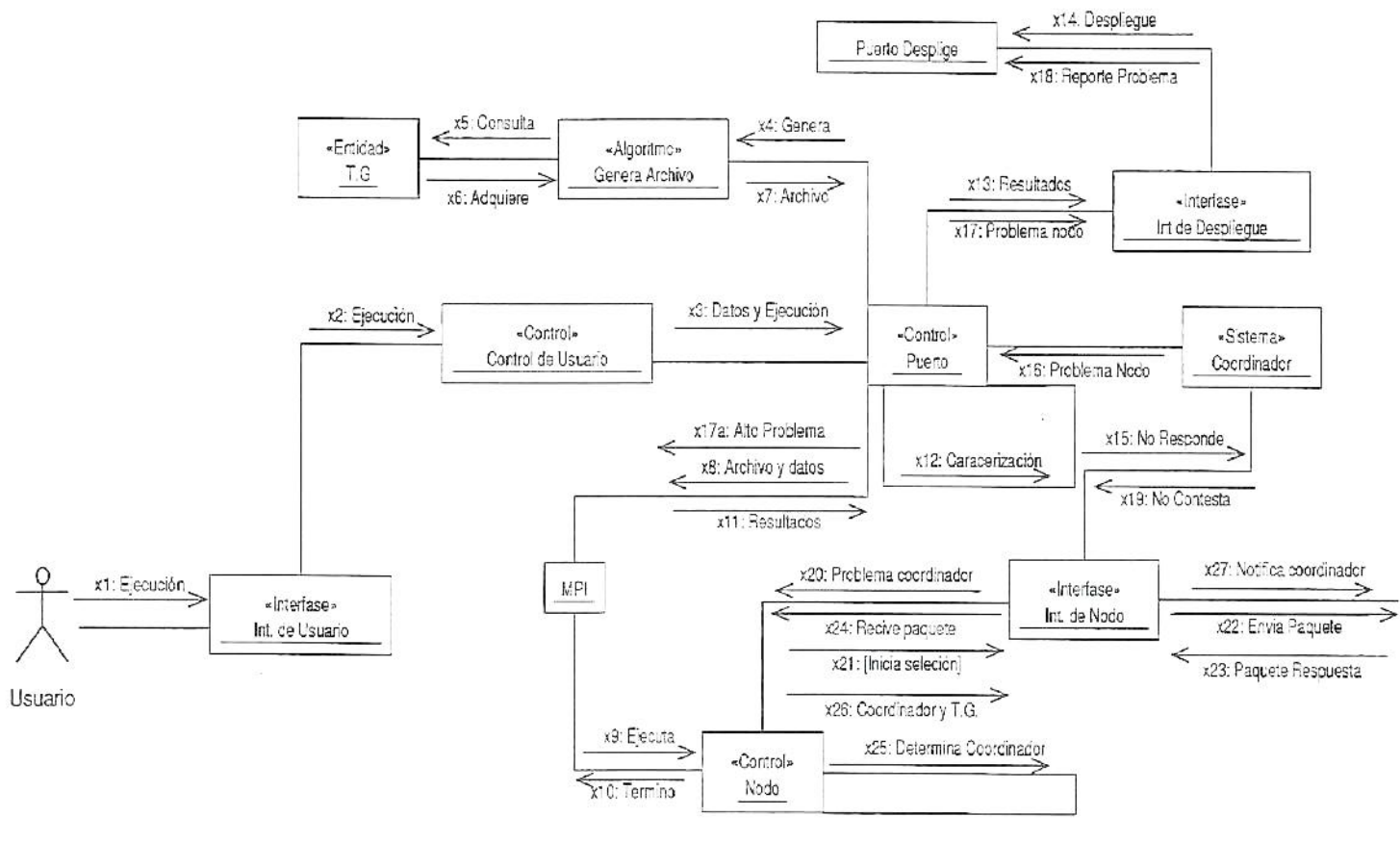
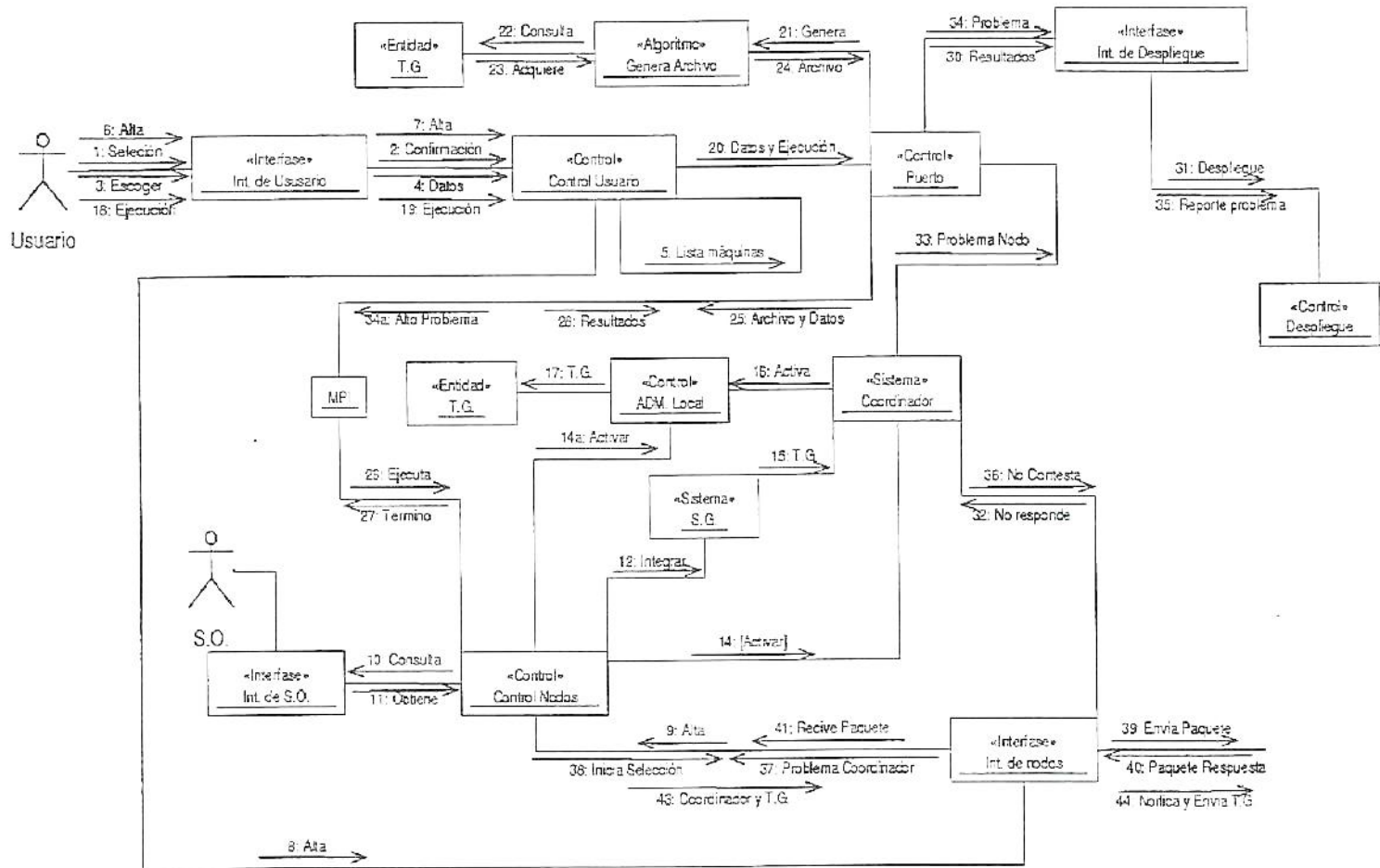


Figura 2.10: Diagrama de Consolidación, Caso de uso Ejecutar y Verificar.



El diagrama de consolidación se divide en seis subsistemas principalmente, esto aplicando los criterios de *Subsystem Structuring Criteria*, y el *Distributed Component Configuration Criteria*, ambos de UML [HA, 2000]. Los criterios se aplicaron con la finalidad de hacerlo configurable. Es decir, darles la oportunidad de ser diseñados como componentes distribuidos que posteriormente puedan ser proyectados a nodos distribuidos. En la figura 2.11, se muestra el diagrama de colaboración de los subsistemas, esto como una vista de alto nivel de la relación que existe entre los subsistemas.

En la figura 2.12, se utilizó agrupamiento de mensajes. Los seis subsistemas que se eligieron son el subsistema **Puerto**, el subsistema **Usuario**, el subsistema **Despliegue**, el subsistema **Nodos**, el subsistema **MPI**, el subsistema **Coordinador**.

El subsistema **Puerto** está compuesto por el control puerto, el algoritmo generador de archivo, y la entidad T.G. En la figura 2.13, se muestra el subsistema de control **Puerto**, para su determinación se aplicaron los criterios de control, del *Subsystem Structuring Criteria* y el criterio de *performance* del *Distributed Component Configuration Criteria*, UML [HA, 2000].

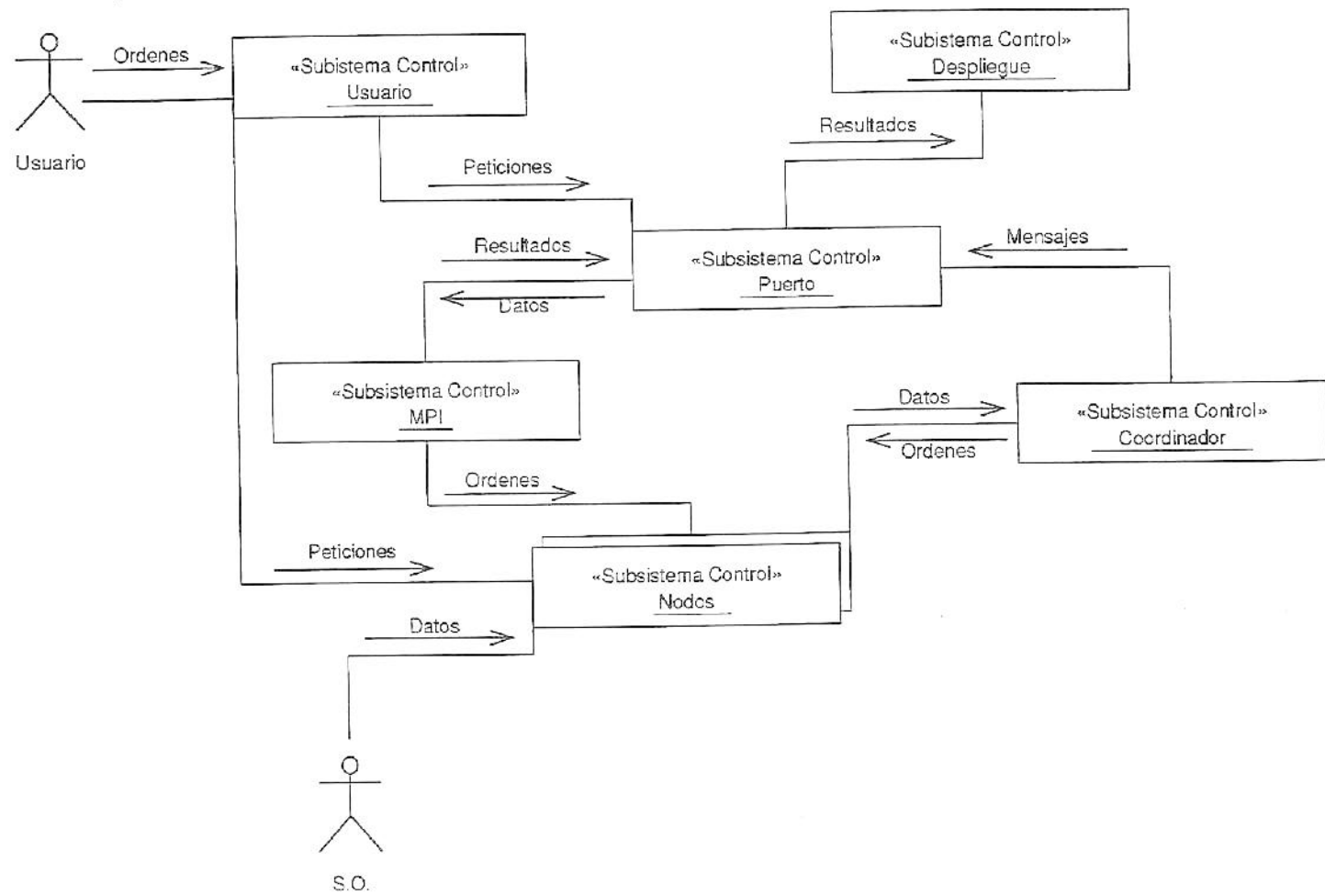
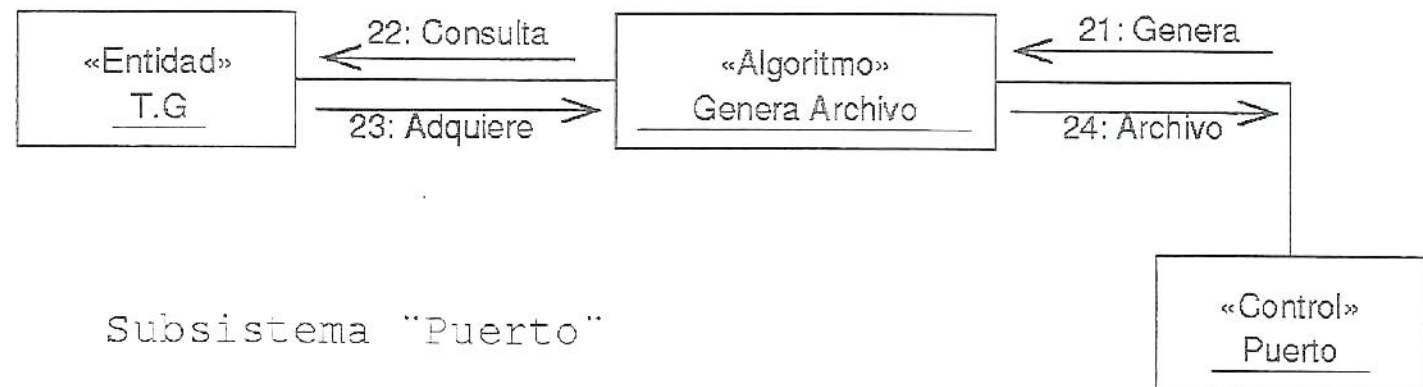


Figura 2.12: Diagrama de Colaboración de Subsistemas.

Figura 2.13: Diagrama de Subsistema Puerto.



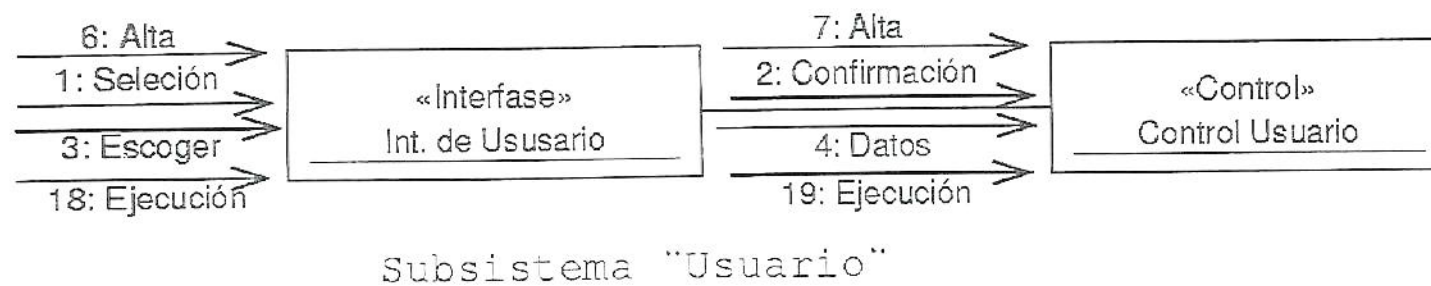
El subsistema **Usuario** está compuesto por la interfase de usuario y el control de usuario como está representado en la figura 2.14, en la cual se aplicaron los criterios mencionados con anterioridad, como se puede apreciar **Usuario** es un subsistema de control y de interfaces.

En los subsistemas de la figura 2.15 sólo se encuentra un miembro dentro del mismo debido a que son subsistemas por sí mismos en la vida real, los miembros son MPI y Coordinador respectivamente.

El subsistema es **Despliegue** está compuesto por el puerto de despliegue, y la interfase de despliegue, esto se muestra en la figura 2.16.

El último es el subsistema **Nodos**, el cual tiene como miembros o está conformado por la administración local, el servidor de grupos, control de nodos, interfase de S.O., interfase de nodos y la tabla de grupos, el subsistema se muestra en la figura 2.17.

Figura 2.14: Diagrama de Subsistema Usuario.



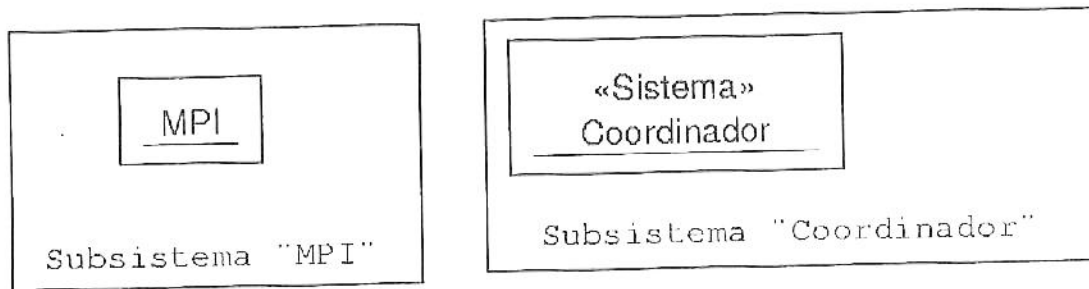


Figura 2.15: Diagrama de los Subsistemas MPI y Coordinador.

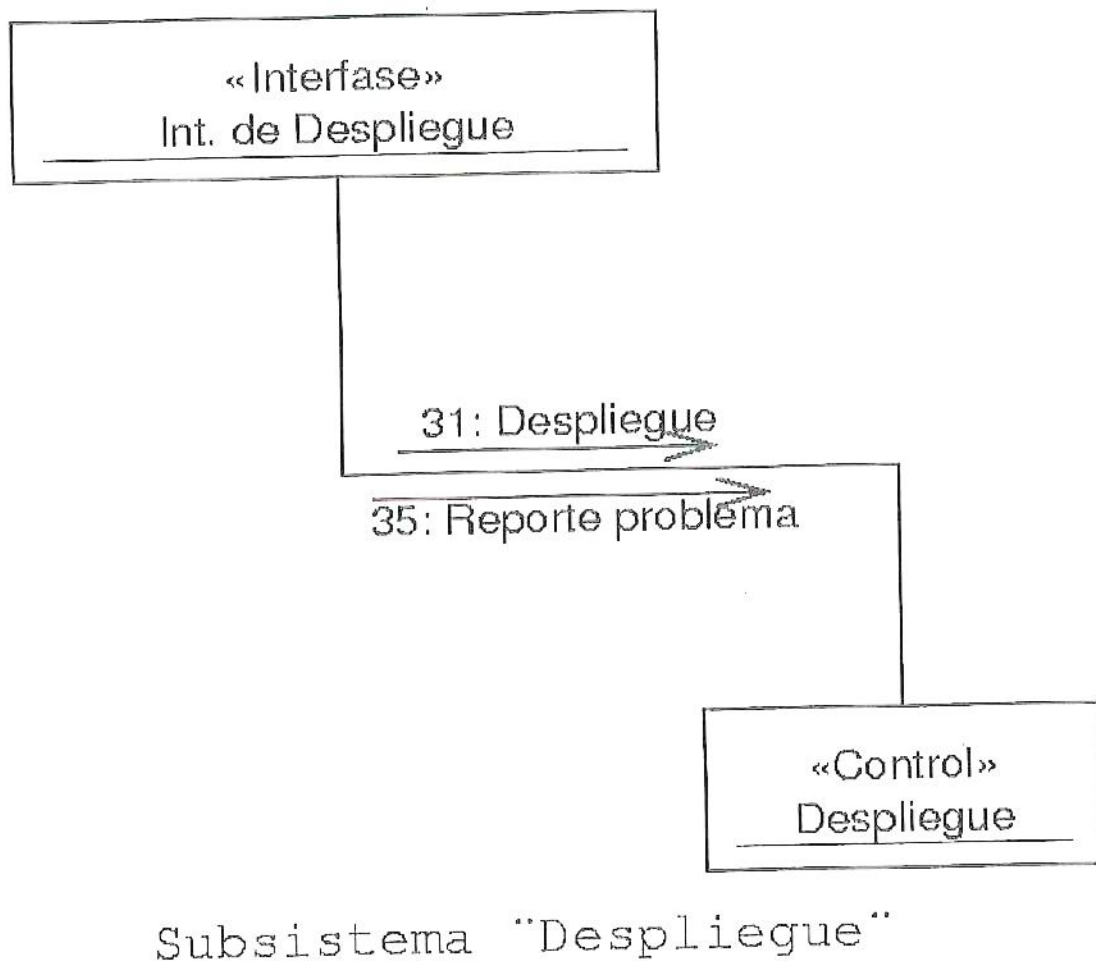
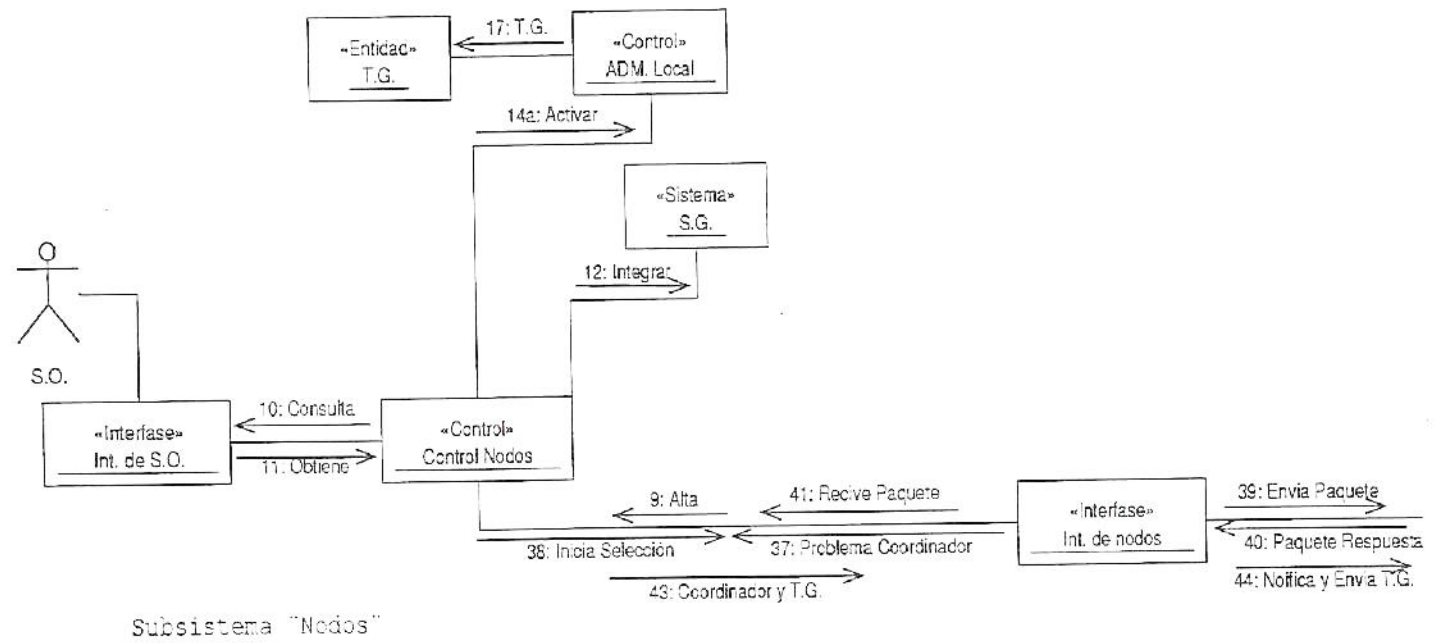


Figura 2.16: Diagrama de Subsistema Despliegue.

Figura 2.17: Diagrama de Subsistema Nodos.



II.8 Diseño.

Para el diseño, en el subsistema **Usuario** se utiliza el esquema orientado a objetos del perl/Gtk, que son un conjunto de bibliotecas orientadas a objetos escritas en C y de libre distribución [GTK+, <http://www.gtk.org>].

En el subsistema de MPI, fue necesario modificar el código fuente de la biblioteca, esto provoca que el esquema original se ve modificado de:

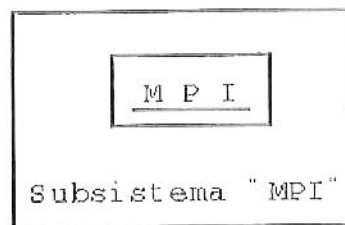


Figura 2.18: Subsistema original *MPI*.

A un subsistema que se usara durante el desarrollo y aplicación de la herramienta, así el subsistema resultante es:

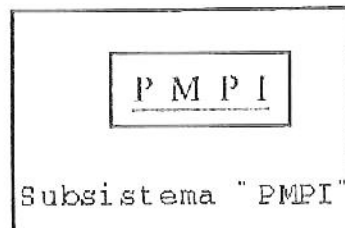


Figura 2.19: Subsistema modificado *PMPI*.

La modificación del subsistema MPI, se realizó mediante a la aplicación de

un lenguaje generador de sobre cubiertas [Capítulo 3].

III Implementación

Introducción

En la actualidad existen muchas maneras de medir el desempeño de los sistemas distribuidos, algunas son usadas al tiempo como criterio de evaluación. Es decir, se toman los tiempos de ejecución y se busca llegar a un tiempo que se considera óptimo, otros buscan aplicar cargas y analizar como se desempeñan en el sistema, estas cargas son denominadas *Benchmarks*. Existe un método que se puede llevar a cabo gracias a las arquitecturas de los microprocesadores. Es decir, consiste en seleccionar y leer los contadores.

Un contador es una parte del microprocesador que mide y reúne eventos relevantes al desempeño en el microprocesador. La cantidad y tipo de eventos que pueden ser medidos varía entre los diferentes microprocesadores debido a que no existen especificaciones en común, y cada fabricante tiene diferentes prioridades en el análisis de desempeño de arquitecturas y de programas. Para esto existen bibliotecas que permiten la lectura y la medición de dichos contadores [Rudolf Berrendorf, 2001].

*PCL*⁵ la biblioteca de contadores de desempeño, establece una plataforma común para la medición de desempeño en un rango extenso de sistemas de

5 Performance Counter Library: Biblioteca de contadores de desempeño.

computadoras. Con una interfase común para todos los sistemas y un grupo definido de eventos orientados a las aplicaciones. Esto da la posibilidad de implementar de una herramienta para la evaluación de desempeño como *PEVTO*⁶. En *PCL* se cuenta con una interfase de bajo nivel que le da al usuario la posibilidad de acceder los registros internos del microprocesador para medir eventos no estándares [Rudolf Berrendorf, 2001].

III.1 Descripción

Como se ha mencionado con anterioridad se utilizan las bibliotecas de *PCL* para obtener el acceso a los contadores y registros del microprocesador aprovechando las ventajas del mismo, entre las cuales están comenzar y detener los contadores, pruebas de funcionalidad y lectura de los valores de los contadores. Así *PCL* soporta llamadas anidadas a sus funciones, lo cual permite obtener medidas de desempeño jerárquicas.

Se realizan dos tipos de experimentos. El primero consiste en la selección de un conjunto de cargas de trabajo o *benchmarks* que se ejecutan de manera no distribuida. Donde sólo se toman en cuenta la función que realiza la actividad o cálculo deseado. El otro esquema de experimentación es el distribuido. Para los programas o *benchmarks* se plantea un método de evaluación o de lectura de los contadores de manera distinta. Es decir, se modificó el código fuente de la

6 Performance Evaluation TOol: Herramienta para la evaluación de desempeño.

biblioteca usada para paralelizar el código, debido a que es por pase de mensajes, esto permite obtener información por cada función o mensaje procesado durante la ejecución del programa, dando la oportunidad de que cualquier código pueda llegar a ser perfilado.

En el desarrollo del proyecto se decidió la utilización de *Linux* como sistema operativo debido al soporte con el que cuenta para la implementación de *clusters*, tanto en software como en la disposición de sus licencias. Así como la facilidad que brinda para realizar modificaciones al *kernel* o núcleo del sistema operativo.

Es necesario la modificación y recompilación del *kernel* o núcleo del sistema operativo *Linux* para poder acceder a los registros y contadores del microprocesador, esto consiste en utilizar el comando propio del sistema operativo *patch*, el cual permite realizar alteraciones a los códigos sólo en las diferencias entre el archivo del "parche" y el código fuente que se desee modificar, esto se puede aplicar a mas de un código fuente a la vez [Free Software Foundation, 1998].

Al momento de compilar el kernel, después de aplicar el parche que da las aptitudes necesarias para acceder a los contadores del microprocesador, basta con activar las opciones de configuración general "*General Setup*", que tiene

como bloque principal *Performance-monitoring counters support*, y dentro de éste, es necesario incluir las opciones *Virtual performance counters support* y *Global performance counters support*. estas opciones deben ser incluidas dentro del kernel. Si se opta por incluirlas como módulos, es necesario tomar en cuenta la configuración dentro del archivo que se encuentra en el directorio */etc/* el nombre de *modules.conf* [Bjorn Ekwall, Keith Owens, 1999]. Por motivos convencionales se decidió incluirlas como parte del *kernel* y no como módulos que pueden ser cargados por él mismo. Esto permite que la imagen generada no realice la fase intermedia de comunicación y control de módulos y así evitar operaciones innecesarias para la lectura de contadores, [Free Software Foundation, 1998].

III.2 Esquemas de la implementación del código.

III.2.1 No distribuido.

El esquema de la evaluación de programas no distribuidos se puede ver de una manera pequeña e ir desglosando su contenido en modelos más completos y un tanto complejos. Esto quiere decir, que es posible tener un esquema del programa en el que se ejecutará el experimento teniendo el desarrollo del programa en una sola función o bloque del programa sin interferir con el resto del código de evaluación. El código para los experimentos no distribuidos se puede ver de manera general como:


```

programa
{
    declaración de variables

    Se inicializa el uso de PCL

    definición de modo de trabajo

    Probar los eventos en la arquitectura

    Se comienza con el conteo

    Función_de_trabajo()

    Se detiene el conteo

    Se sale del uso del PCL.
}

```

Como se muestra es posible construir un cuerpo común en código y aislar el problema que se está analizando en una sola función, la cual se analizará durante la ejecución.

La **declaración de variables**, es donde se declaran los arreglos y variables necesarios para llevar a cabo la manipulación de contadores. Es necesario declarar un arreglo del tipo de contadores *PCL*, donde se guardarán los resultados de los contadores cuando el número del evento está en rango inferior al de los contadores de operaciones punto flotante, el arreglo debe de ser de dos localidades. Se requiere de un arreglo de dos localidades del tipo

contadores de punto flotante de *PCL*, lo anterior se puede expresar en el código de la siguiente manera.

```
PCL_CNT_TYPE i_result[2] = {0,0};  
PCL_FP_CNT_TYPE fp_result[2]={0,0};
```

Para los experimentos no distribuidos el arreglo puede ser sustituido por una variable que debe de ser de tipo *PCL_CNT_TYPE* debido a que los contadores son números enteros de 64 bits. Los contadores soportados por *PCL* son 60 pero los contadores soportados por la plataforma Intel y que se encuentran dentro de *PCL* son 22, en lo que es Pentium MMX, Pentium Pro, Pentium II, Pentium III y Pentium IV, [Anexo 2].

La manera de manejar los contadores dentro del código puede ser la siguiente.

```
int counter_list[2];
```

La razón por la que el arreglo debe de ser de 2 localidades es por que la arquitectura Intel sólo soporta la medición de 2 contadores a la vez [http://www.intel.com, 2001]. Es necesario incluir una variable de tipo entero sin signo para contener el modo de trabajo, en *PCL* existen 3 modos de trabajo, el

modo **usuario**, **sistema** y **usuario-sistema**. La diferencia entre los modos de trabajo aparece en el anexo 2, la manera en que se puede manejar la bandera para el modo de trabajo dentro del código es como sigue.

```
unsigned int flags;
```

Y un descriptor de hilos para el *PCL*, que se puede manejar así.

```
void *descr;
```

Al *iniciar el uso del PCL*, se aloja un descriptor de hilos específico del cual debe pasarse su dirección a las demás funciones del *PCL*, el proceso de inicializar se lleva acabo mediante el uso de la función *PCLinit* y el uso del descriptor es como se muestra a continuación.

```
PCLinit(&descr)
```

La *definición del modo de trabajo*, se lleva a cabo con el uso de la bandera previamente mencionada.

Se *prueban los eventos en la arquitectura*, de esta forma podemos estar

seguros que los eventos seleccionados trabajan de manera satisfactoria en nuestra arquitectura, esto se maneja con el uso de la función *PCLquery* la cual prueba por funcionalidad si ésta se encuentra disponible. Por ejemplo.

```
PCLquery(descr, counter_list, 1, flags)
```

Al **comenzar con el conteo**, se inicializa con el evento si es posible, usando los tipos que se encuentran en el arreglo `counter_list[2]`, la manera de usar esta función es definida `counter_list[ ]`.

```
PCLstart(descr, counter_list, 1, flags)
```

Posteriormente, se comienza a ejecutar la tarea o carga de trabajo que se desea medir. Para el caso específicamente de los experimentos realizados en el análisis de *PEVTO* se usan funciones donde se encuentra el código del trabajo a evaluar, así se puede manejar un mismo orden en el desarrollo de los experimentos, dando la ventaja de que rápidamente se puede adaptar el fuente a nuevos experimentos sin la necesidad de tener que escribir todo el programa para cada una de las cargas o hacer grandes cambios en ellos, ya que podría ser un problema de evaluación muy largo o extenso.

El **detener el conteo**, es cuando se detiene el conteo de desempeño y se

regresan los valores de los contadores, los resultados son puestos o colocados en el arreglo provisto por el usuario, en este caso, *i_result*.

Por último se requiere **salir del PCL** para poder liberar el descriptor específico del hilo, para esto solo falta usar la función *PCLexit*.

PCLexit(descr)

III.2.2 Distribuido.

El esquema de experimentos distribuidos es mas complejo debido a que para realizar el análisis de desempeño no basta con medir o perfilar la tarea o trabajo que se necesite ejecutar. Además es necesario conocer los datos referentes a todas la funciones, llamadas o interrupciones usadas por el paso de mensajes en la paralelización, ya que este fue el método usado dentro de la experimentación distribuida, en específico se utilizó *mpich*⁷.

Para poder obtener información de todas y cada una de las funciones o interrupciones posibles dentro de *mpich* fue necesario modificar el código fuente de las bibliotecas estándares de paso de mensajes, Para modificar las bibliotecas se requirió utilizar el generador de código *wrappergen*. El *wrappergen* se encuentra incluido dentro del *mpich* en la sección de *MPE* o

⁷ Mpich, del MPI Message Passing Interface.

mpich extensions. El *wrapper* proporciona un grupo de etiquetas y bloques que permiten definir y controlar la manera en que las funciones de las bibliotecas son manipuladas [Antony Chan, William Gropp and Ewing Lusk, 1998].

El generador de sobrecubiertas de perfilado (*wrappergen*) ha sido diseñado para complementar a la interfase de perfilado del *MPI*. Eso permite al usuario escribir un gran número de pseudo "*wrappers*" que pueden ser aplicados a cualquier número de funciones de *MPI*.

Wrappergen necesita tres tipos de entradas fuentes:

1. Una lista de funciones para las cuales generar las sobrecubiertas.
2. Declaraciones de las funciones que serán perfiladas. Esto puede ser encontrado por omisión en el archivo ***mpi_proto*** para las funciones de *MPI-1* y para las funciones de Entrada y Salida se encuentran en el archivo ***mpiio_proto***.
3. Definiciones de sobrecubierta.

La lista de funciones es un simple archivo de texto con los nombres de las funciones separados por espacios. Si no se proporciona ningún prototipo de las funciones, las que se encuentran en ***mpi_proto*** y ***mpiio_proto*** serán utilizadas, [Antony Chan, William Gropp and Ewing Lusk, 1998].

Las opciones para el *wrappergen* son:

1. **-w archivo:** adhiere el nombre del fuente a la lista de archivos de sobrecubierta que será utilizado.
2. **-f archivo:** contiene la lista de nombres de las funciones que serán perfiladas separadas por espacios.
3. **-p archivos:** contiene las declaraciones de los prototipos de las funciones especiales.
4. **-o archivo:** manda la salida al archivo.

III.3 Escribiendo definiciones de sobrecubierta.

Las definiciones de sobrecubierta consiste en código de C con *macros* especiales. Cada *macro* se encuentra rodeado con `{{}}`. Para la experimentación se utilizó solo una parte de los *macros* provistos por *PCL*, debido a que para el caso específico de *PEVTO* no fue necesaria la utilización de todos ellos [Antony Chan, William Gropp and Ewing Lusk, 1998]. Una de las principales definiciones del *wrappergen* es:

```
{{fnall <nombre de función> <función A> <función B> ...}}  
  
...  
  
{{callfn}}  
  
...  
  
{{endfnall}}
```


{{fnall}} define un macro que será utilizado en todas funciones excepto en las funciones listadas dentro del mismo *macro*. Esto se expandirá a toda una definición de función en formato de C. El *macro* **{{callfn}}** indica al *wrappergen* donde insertar la llamada a la función que está siendo perfilada, en este caso debe de haber por lo menos una llamada a **{{callfn}}** en cada definición de sobrecubierta.

```
{{fn <nombre de función> <función A> <función B> ...}}  
  
...  
  
{{callfn}}  
  
...  
  
{{endfn}}
```

{{fn}} es idéntico a **{{fnall}}** excepto que sólo genera definiciones de sobrecubierta para las funciones listadas dentro del *macro* [Antony Chan, William Gropp and Ewing Lusk, 1998].

III.4 Generando la biblioteca.

Para facilitar el trabajo y automatizar la ejecución de los códigos distribuidos se generó un nuevo código fuente de la biblioteca del *Mpich*, para lo cual se utilizó el *wrappergen*. En base al código generado se obtuvieron los códigos objetos. Es decir, al compilar no se efectúa la etapa de enlace, es solo código

arrojado por el ensamblador.

Este código es agrupado en la biblioteca gracias al archivador *ar*, el cual crea, modifica y extrae de archivos. El archivo resultante simplemente contiene una colección de otros archivos en una estructura que hace posible recuperar los miembros de manera individual.

Las opciones utilizadas como parámetros para generar las bibliotecas son *c*, *u* y *r*, las cuales permiten.

c Crear un archivo. El archivo es creado si no existe, cuando se solicita una actualización.

u Se utiliza solamente para agregar los archivo que son nuevos en comparación con los que ya se encuentran, esta operación sólo se puede utilizar con *r* (reemplazar).

r Inserta los archivos miembros con reemplazo.

Al agrupar los archivos objetos dentro de la biblioteca es necesario generar el índice interno de la biblioteca, esto es posible son *ranlib* la cual permite generar un índice del contenido de un archivo. El índice guarda la lista de cada símbolo definido por cada uno de los miembros de un archivo. EL archivo es un código objeto movido de lugar [Free Software Foundation, Inc, 2002].

III.5 Interfase.

La herramienta para la evaluación de desempeño, tiene un ambiente integrado de trabajo, el cual se divide en tres partes, la pantalla principal, el área experimentos distribuidos y el área de experimentos no distribuidos. Todas y cada una de las partes que forman la herramienta y la interfase se encuentra codificadas en *perl* y *GTK*, [Anexo código].

III.5.1 Pantalla principal.

La ventana principal es una pequeña interfase la cual coordina las otras actividades de la herramienta, consta de dos menus y una pequeña imagen con el logo de la aplicación, como se muestra en la figura 3.1.

Para crear la venta, se utilizó la llamada de *GTK* para una ventana de alto nivel, lo cual permite el libre diseño de la ventana sin importar el tipo o cantidad de objetos que contenga. La llamada que se utilizo fue.

```
$ventana = new GTK::Window( "loplevel" );
```

Así de esta manera la variable *\$ventana*, o widget como le llamaremos en adelante a las variables que contengan objetos de la interfase, obtiene todas las propiedades de una ventana de alto nivel, así como sus señales, eventos y el control del bucle de ejecución de todo el *GTK*, [Anexo código]



Figura 3.1: Pantalla principal de la interfase de *PEVTO*.

El menú "Experimentos", es el que controla las opciones de "Distribuidos" y "No Distribuidos", así como la opción de "Salir". El primer elemento del menú controla todo lo referente a la selección y generación de aplicaciones y cargas distribuidas para la evaluación. La segunda opción se refiere a las cargas que no se ejecutan de manera distribuida. Los menus principales y sus opciones son definidos en un arreglo para después ser creados con la llamada a función, figura 3.2.

```
$item_factory->create_items( @cosas_menu );
```

Donde @cosas_menu, es el arreglo que contiene todos los menús y sus elementos, así como las opciones y funciones que deben de ser activadas al seleccionado. \$item_factory, es el widget que contendrá al menú, [Anexo código].



Figura 3.2: Menú principal.

El segundo menú "Ayuda", presenta dos opciones "Manual" y "Acerca de ...". La primera da un manual sobre como usar la aplicación y como son los contadores del microprocesador. La segunda da una descripción sobre la licencia de distribución de la herramienta y los datos del autor [Anexo código], como se muestra en la figura 3.3.



Figura 3.3: Menú de ayuda.

III.5.2 Experimentos distribuidos.

La segunda parte de la herramienta es el área de experimentos distribuidos, aquí la primer ventana es donde se selecciona la aplicación distribuida que se desea ejecutar. Las aplicaciones son leídas de un directorio, cualquier aplicación que sea distribuida y que desee perfilar solo es necesario copiarla dentro de este directorio y el sistema automáticamente lo agregará a la lista de aplicaciones, las aplicaciones deben estar en código fuente. Es decir, sin compilar. Esta es una ventana de alto nivel pero solo posee el control del bucle de ejecución sobre sí misma, y es creada con la llamada a la función.

```
$ventana = new Gtk::Window( "toplevel" );
```

y se le asigna su propio bucle de control

```
$ventana->signal_connect( "destroy", sub { $ventana->destroy(); } );
```

Cabe mencionar que todas las funciones y llamadas sobre cargas distribuidas se encuentran en el módulo *cargas_distribuidas.pm*, es por eso que las variables correspondientes a cada ventana se declaran de manera local, [Anexo código]. Como se muestra en la figura 3.4.



Figura 3.4: Selección de carga distribuida.

Los botones que se encuentran en la ventana fueron agregados como botones normales con etiquetas.

```
$boton_a = new Gtk::Button( " Agregar " );  
  
$boton = new Gtk::Button( " Siguiente -> " );  
  
$boton_cerrar = new Gtk::Button( " Cerrar " );
```

La ventana tiene una sección de selección de archivos, y tres botones los cuales tienen su función específica, el primer botón "Agregar", permite seleccionar el archivo que se deseen incluir al directorio de los programas distribuidos, así estará disponible para ser utilizado en otras ocasiones. Para este caso se usa la siguiente llamada.

```
$file_dialog = new Gtk::FileSelection( "Selección de archivos" );
```

Como se muestra en la figura 3.5.

El segundo botón "Siguiente ->", lleva a la ventana en la que se selecciona el par de contadores de desempeño para el experimento. Los contadores son necesarios para poder generar correctamente el ejecutable del experimento, en la lista se encuentran todas las posibles combinaciones de contadores, en la mayoría de ellos no importa el orden en el que son leídos, pero en los contadores que aparecen al final de la lista sí importa el orden. La ventana contiene una lista la cual es colocada dentro de una venta de alto nivel. La llamada necesaria para crear la lista es:

```
$clist = new_with_titles Gtk::CList( @ titulos );
```

Esto se puede apreciar en la figura 3.6.

La opción de siguiente nos lleva a la ventana donde se pueden introducir información referente a la biblioteca necesaria para la construcción del ejecutable que usaremos como carga de trabajo. La última ventana es de alto nivel, contiene los botones "Siguiente ->" y "Cancelar", como se muestra en la figura 3.7.

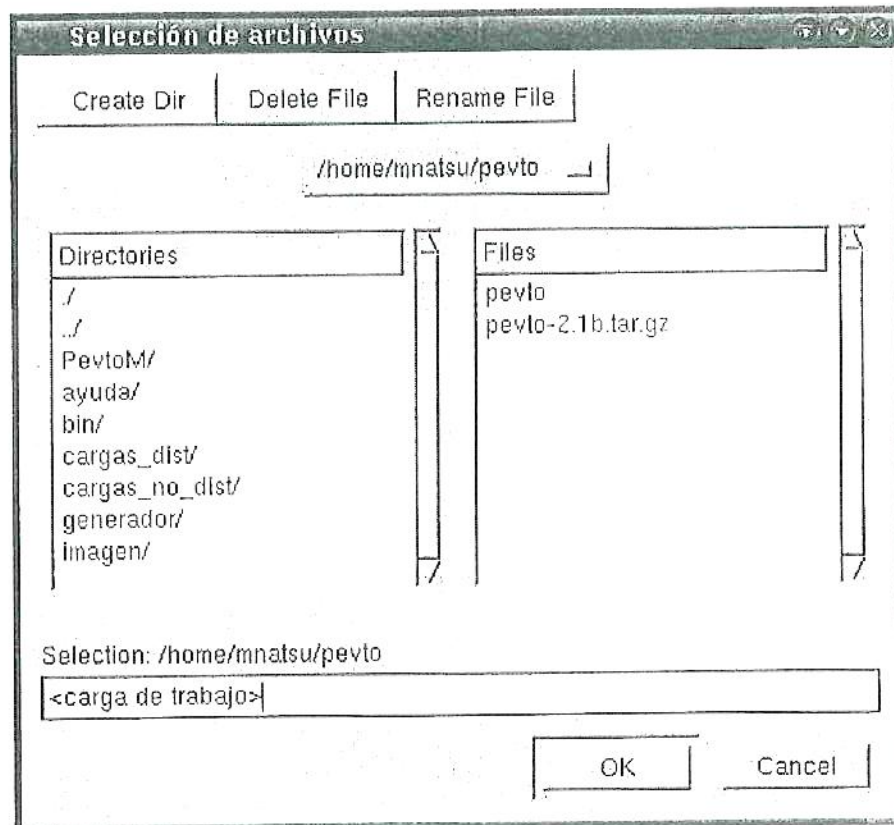


Figura 3.5: Ventana de selección de archivos.

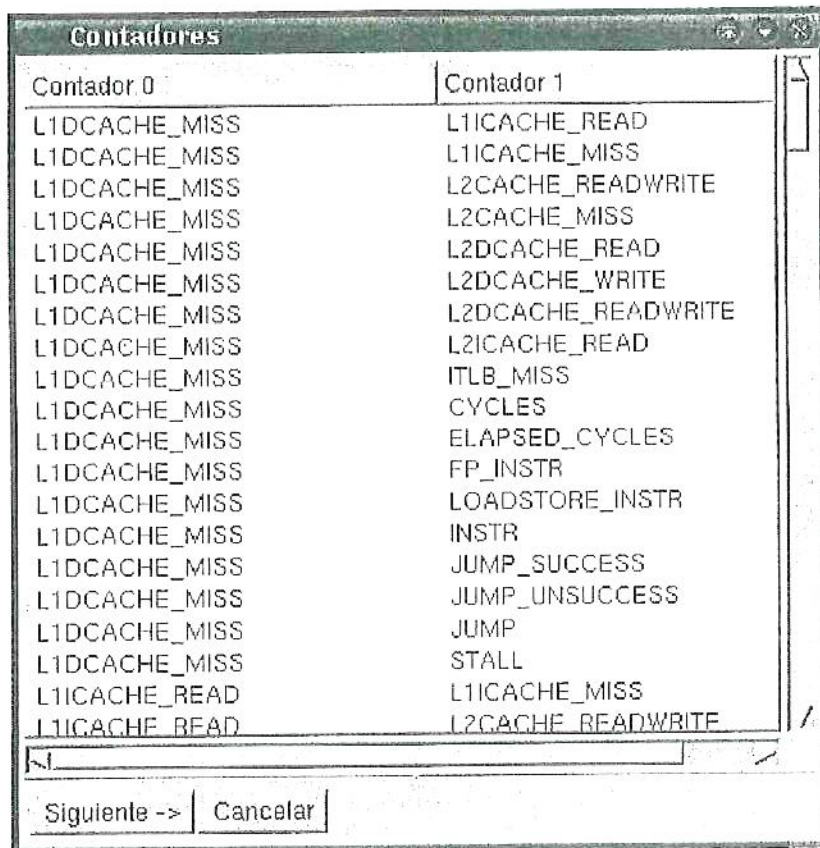


Figura 3.6: Ventana de contadores.

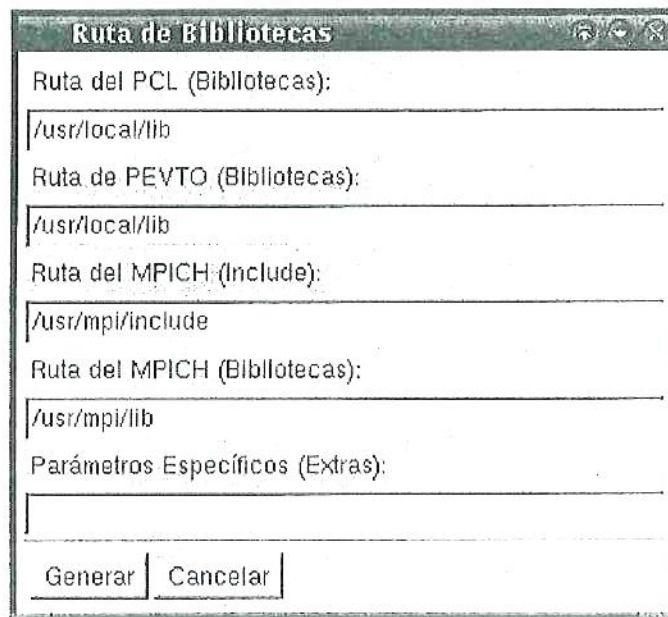


Figura 3.7: Ventana de ruta de bibliotecas.

III.5.3 Experimentos no distribuidos.

La tercer parte de la herramienta es el área de experimentos no distribuidos, aquí la primera ventana es en donde se selecciona la aplicación no distribuida que se desea ejecutar, esta es una ventana de alto nivel y se genera de la misma manera que se generan las anteriores, figura 3.8.

```
$ventana = new Gtk::Window( "toplevel" );
```

Una de las opciones más importantes de esta sección, es la opción de "Generar", la cual da una ventana de selección de archivos, la cual permite seleccionar una carga no distribuida en un formato definido [Anexo, Manual de

usuario], y generar un nuevo código perfilado y listo para compilar así como agregarlo al directorio de donde se seleccionan las cargas no distribuidas, así si se desea agregar diferentes cargas esta opción da la libertad de hacerlo [Anexo, Manual de usuario], figura 3.9.

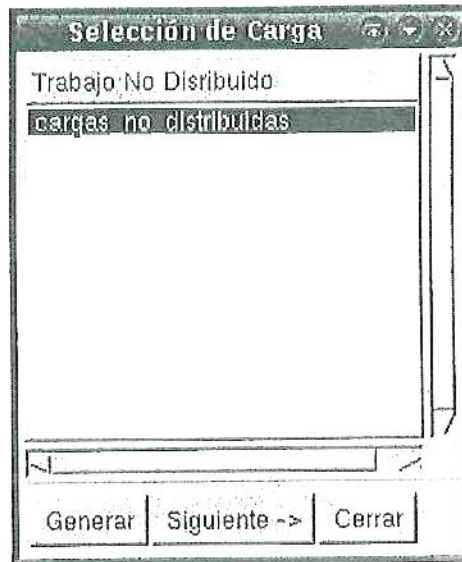


Figura 3.8: Selección de cargas no distribuidas.

La última de las pantallas presentadas dentro de las cargas no distribuidas es la selección de la ruta de las bibliotecas *PCL*, debido a que la aplicación se basa en la utilización de la biblioteca de contadores de desempeño, y da la opción de generar el ejecutable [Anexo, Manual de usuario], figura 3.10.

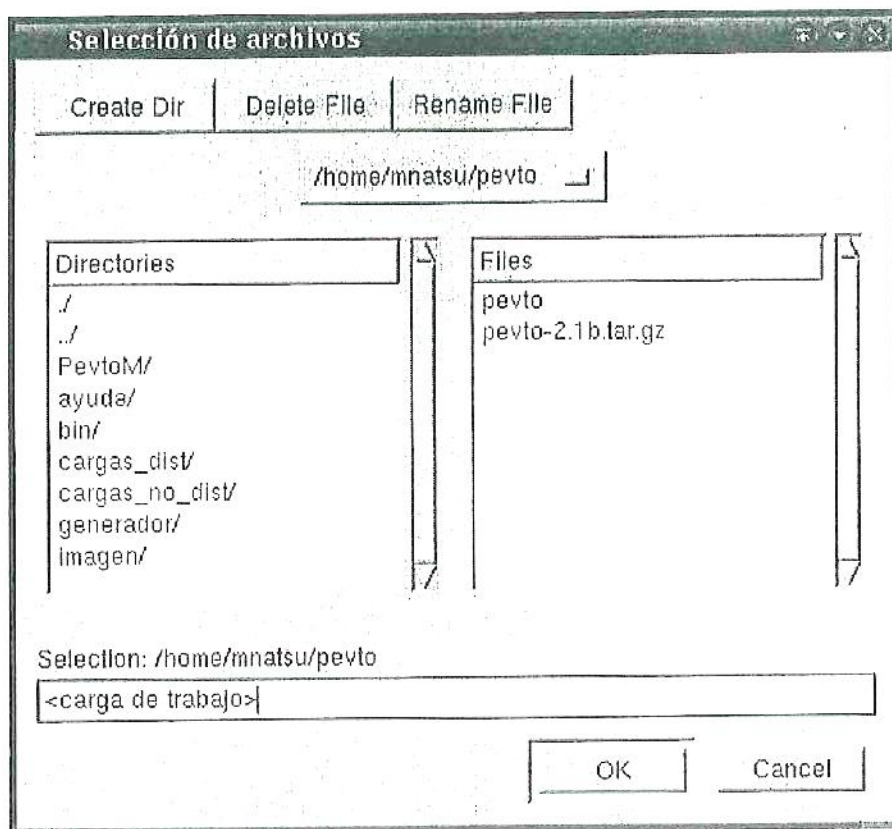


Figura 3.9: Selección y generación de cargas no distribuidas.

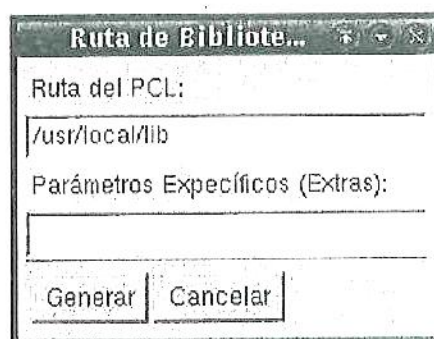


Figura 3.10: Ruta de bibliotecas PCL.

IV Análisis Experimental

Introducción.

En este capítulo se presentan los experimentos y resultados de la implementación de diversas cargas de trabajo en paralelo, obteniendo con esto las lecturas de los contadores de desempeño del microprocesador. La información recabada es analizada por el modelo matemático del punto medio operacional para generar información experimental que describa el desempeño del sistema observado. Los resultados son vistos en forma de vectores denominados *vectores de desempeño* [Anexo x].

En los experimentos se seleccionaron 6 cargas de trabajo con características distintas entre ellas, se tomaron en cuenta las características que generan en la máquina o sistema para poder llevarse a cabo. Es decir, se tomó en cuenta la cantidad de accesos a memoria, los millones de operaciones de punto flotante que realiza, los accesos a memoria *cache*, cantidad de ciclos que hace, etc. Esto con la finalidad de probar la lectura de distintos contadores de desempeño.

En la implementación de los experimentos fue necesario modificar las bibliotecas de paso de mensajes, en particular *MPI (mpich)*, para la lectura de los contadores del microprocesador, lo cual presento el problema de ser

dependiente de la arquitectura, esto limitó la manera en como se obtiene la información. La plataforma seleccionada para la experimentación fue la *Intel PIII*, que sólo permite leer dos contadores de manera simultánea y existen contadores que deben leerse en un orden específico bajo ciertas circunstancias [Anexo, *PCL*].

La herramienta da la posibilidad de generar cargas de trabajo distribuidas y no distribuidas, en la presente era necesario evaluar un sistema de imagen única, por lo que en la experimentación sólo se consideran los experimentos distribuidos.

La experimentación puede ser llevada a cabo, tomando en cuenta las limitaciones de la plataforma seleccionada y generando nuevas bibliotecas para el paso de mensajes, estas bibliotecas deben de ser usadas para generar cargas de trabajo capaces de leer información de los registros del microprocesador.

IV.1 Descripción de las cargas de trabajo.

Para realizar los experimentos se utilizaron 6 programas que son: Estimar el valor de Pi , Fractales de *Mandelbrot*, Matriz transpuesta, Integral del $\cos(x)$, la ecuación de Laplace usando el método de iteraciones de Jacobi y la ecuación de Laplace usando el método sucesivo sobre relajación y aceleración de

Chebyshev. Tales se describen en los siguientes párrafos.

IV.1.1 π .

El primer experimento es un programa que calcula el valor aproximado de π , usando un esquema numérico, como se encuentra en la figura 4.1.

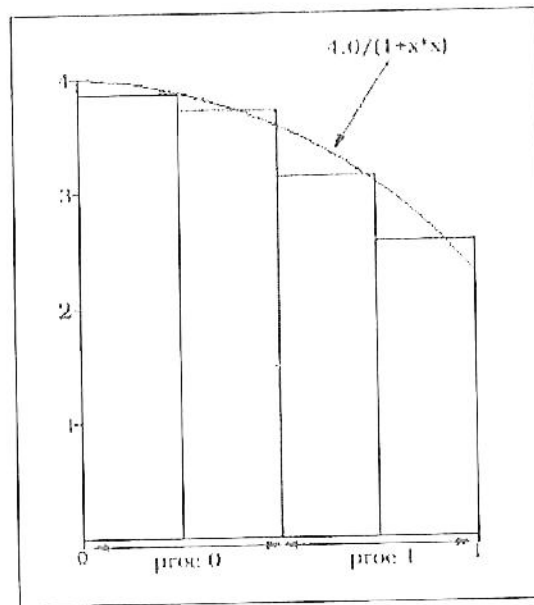


Figura 4.1: Representación de π .

De acuerdo a la figura 4.1, el valor de π puede ser aproximado como la suma del área de los N rectángulos, donde $N=4$ para el diagrama anterior. La figura muestra como la carga de trabajo es distribuida a través de los procesadores (para los 2 procesadores en este caso particular). En la implementación paralela cada proceso computa el área del rectángulo que está en su dominio, seguido de una suma global para poder aproximar el valor de π ,

[Neil Matthew y Richard Stones, 2001].

Con base en el funcionamiento e implementación, este experimento presenta un alto grado de accesos a memoria, realiza muchas instrucciones, accesos a los diferentes niveles de cache, y presenta una paralelización por datos. Es decir, se hacen las mismas operaciones a diferentes datos. Esto fueron algunos de los criterios considerados para tomar a *Pi* como una de las cargas de trabajo para los experimentos.

IV.1.2 Mandelbrot.

El segundo experimento tomado es los fractales de Mandelbrot, aquí se buscan los valores de c que dan solución a la ecuación iterativa compleja, $z = z + z^2 + c$ comenzando de $z = 0$. Se puede ver que si el módulo de z se vuelve mayor que 2 cuando esta iterando, entonces z crecerá sin límites dando una solución inestable. Esta regla es usada para calcular una solución aproximada del conjunto de Mandelbrot iterando la ecuación un número específico de veces para cada c . Cada valor de c para el cual el módulo de z permanezca menor a 2 será parte del conjunto de Mandelbrot y asignando un color proporcional al número de iteraciones toma para el módulo de z para incrementar más allá de 2 el resto de los puntos.

Mandelbrot es computado de un grupo de puntos dentro de una región

cuadrada de 4 de longitud centrada en el origen del plano complejo. El programa paralelo de Mandelbrot usa un esquema de proyección en donde la reja es dividida en regiones rectangulares paralelas al eje y y asignada al procesador. La única interacción entre los procesadores en este programa es cuando se reúne el arreglo distribuido entre los procesadores para formar el arreglo de la imagen final [Neil Matthew y Richard Stones, 2001].

Las operaciones que realiza el programa que calcula los conjuntos de Mandelbrot son: operaciones de punto flotante, accesos de cache, instrucciones realizadas, operaciones con números imaginarios, son de gran utilidad para medir y leer los contadores del microprocesador en distintos aspectos como los previamente mencionados.

IV.1.3 Matriz Transpuesta.

En el siguiente experimento consideramos un programa para hacer la transpuesta de una matriz. La transpuesta de la matriz es una operación importante en muchos algoritmos científicos. El método de particionar una matriz entre los procesadores es usualmente determinado por el comportamiento de la comunicación de todo el programa. En nuestro caso utilizamos partición orientada a bloques de 1-D de la matriz a los procesadores como se muestra en la figura 4.2.

En la figura 4.3, se ilustra el algoritmo usado para la transformación de la matriz. Si la matriz es de dimensiones $N * N$ y si hay M procesadores con N exactamente divisible entre M , entonces la matriz es dividida en $M \wedge 2$ sub matrices de dimensión $(N/M) * (N/M)$. La figura 4.3 muestra una matriz de 4x4 y de dos procesadores. Cada procesador tiene M sub matrices. A_{ij} Denota la j -esima submatriz del procesador i . Cuando se transpone la matriz, el procesador i cambia su sub matriz j a la sub matriz i del procesador j .

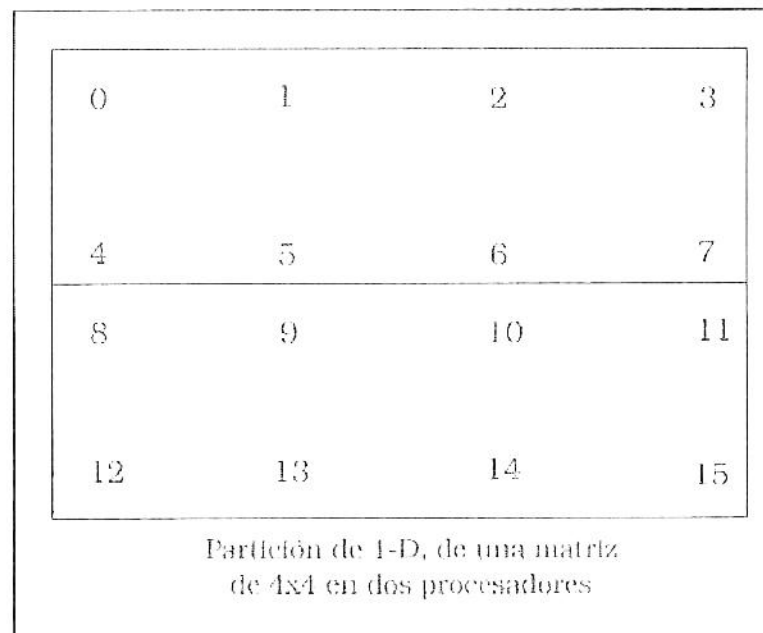


Figura 4.2: Partición en 1-D de una matriz de 4x4.

El obtener la matriz transpuesta puede parecer un procedimiento sencillo de realizar, pero si se considera una matriz muy grande. Es decir, una matriz con

muchos datos, esto puede llegar a ser una carga considerable para el sistema, tanto en el control de su *cache* como en el control de la paginación de la memoria. Esto permite que se considere a la matriz transpuesta como una carga de trabajo útil, ya que brinda la posibilidad de obtener lectura en los contadores de control de memoria, ciclos concluidos, instrucciones realizadas y casi todos los contadores de control de cache.

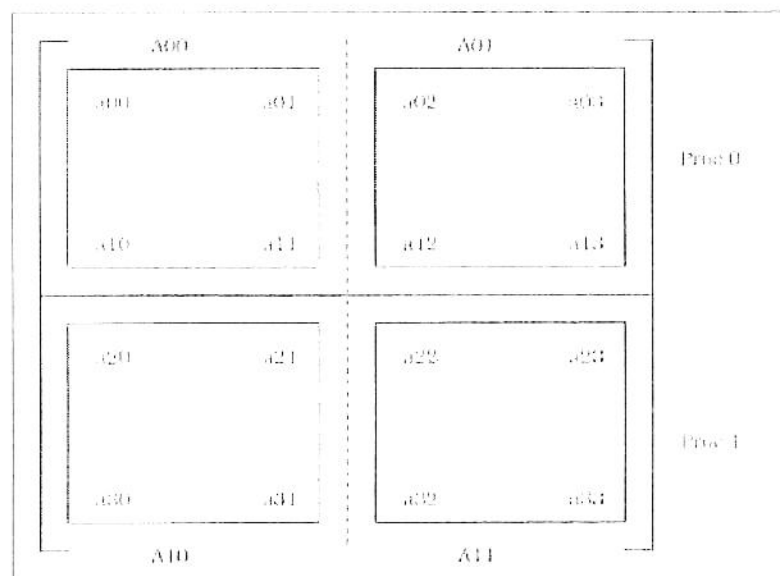


Figura 4.3: División de j matrices en i procesadores.

IV.1.4 Integral $\cos(x)$.

En este programa se pretende utilizar el código que resuelve la integral acotada de $\cos(x)$ por el método del punto medio, dentro de límites definidos. Tomando en cuenta que una integral puede ser expresada en términos de sumatorias se hace lo siguiente:

$$\int_a^b \cos(x) dx = \sum_{i=0}^{p-1} \sum_{j=0}^{n-1} \int_{aij}^{aij+h} \cos(x) dx \simeq \sum_{i=0}^{p-1} \left[\sum_{j=0}^{n-1} \cos\left(aij + \frac{h}{2}\right) \cdot h \right]$$

Donde p es el número de procesadores, n es el número de intervalos por proceso, a es el límite inferior de integración, b es el límite superior de integración, $h = (b - a) / (n \cdot p)$, $aij = a + [i \cdot n + j] \cdot h$. Con esto definimos lo que vamos a calcular que son las áreas de los rectángulos expresados en la figura 4.4.

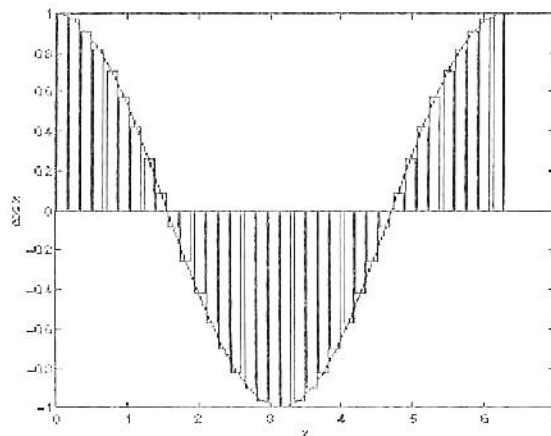


Figura 4.4: Integral del $\cos(x)$.

Tomando esto, se desarrolló un programa en paralelo capaz de resolver la integral de $\cos(x)$, esto permite obtener lecturas de contadores una gran diversidad de ellos, como son: escritura en el cache de datos, lecturas del *cache* de datos, escrituras en el *cache* de datos, en las instrucciones de punto flotante,

ciclos de reloj, lecturas y escrituras al *cache* de instrucciones, ciclos de reloj transcurridos, etc. Así la integral resulta ser una carga de mucho interés en la realización de nuestro trabajo.

IV.1.5 Laplace.

En el caso de la transformada de Laplace, se consideraron dos maneras de resolverla que presenta características computacionales distintas, para poder entender el comportamiento de esta carga trabajo es necesario definirla para poder entender su comportamiento y las partes que la conforman. La ecuación de Laplace se define de la siguiente manera: si $f(x)$ está definida cuando $t \geq 0$, la integral impropia:

$$\int_0^{\infty} K(s, t) \cdot f(t) dt$$

Se define como un límite: $\int_0^{\infty} K(s, t) \cdot f(t) dt = \lim_{b \rightarrow \infty} \int_0^b K(s, t) \cdot f(t) dt$. Si

existe el límite, se dice que la integral existe o que es convergente; si no existe el límite, la integral no existe y se dice que es divergente. En general, el límite anterior existe sólo para ciertos valores de la variable s . La sustitución

$K(s, t) = e^{-st}$ proporciona una transformación integral.

Al desglosar las características es fácil entender el comportamiento y considerar sus características al momento de la paralelización, los métodos usados para resolver esta ecuación son el método de iteraciones de Jacobi y la ecuación de Laplace usando el método sucesivo sobre relajación y aceleración de Chebyshev. Ambos métodos son muy iterativos, es decir presenta una gran cantidad de accesos a memoria, fallos de caches, y unos de ellos el de Chebyshev agrega ordenación y otros aspectos que generan trabajo de manera considerable en la máquina, esto dependerá del número de iteraciones que se utilicen para resolverlas.

IV.2 Ejecución de los experimentos.

Los experimentos antes mencionados fueron compilados con distintas bibliotecas, de manera que se pudiera obtener información sobre distintos contadores del sistema para cada ejecución de las cargas mencionadas. Las características de las ejecuciones son como se muestra.

El ambiente para los experimentos consta de:

- Una arquitectura de imagen única tipo cluster, de 3 nodos con 2 procesadores *Intel PIII*, 128Mb de RAM, 8Gb, Tarjeta Ethernet de 10/100Mbits.
- *Switch "NetGear"* ethernet de 10/100Mbits.
- *Mpich*, version 1.2.4, para el paso de mensajes.

- *PCL*, version 2.1, para el acceso a los registros del microprocesador.
- Compilador de C, *GNU* version 2.96.
- *Perl*, version 5.6.1.

En el proceso de compilación de los experimentos, en específico en la etapa de enlace se incluyeron las bibliotecas generadas para el acceso a los distintos contadores, teniendo que repetir este proceso varias veces para una misma carga de trabajo, esto con la finalidad de generar distintos códigos ejecutables, uno por cada contador que se desee evaluar.

IV.3 Resultados.

Los resultados obtenidos del proceso de experimentación, son lecturas de los registros del microprocesador, esta información esta expresada a nivel proceso. Es decir, se obtiene el tiempo y la lectura de los contadores por cada proceso que participó en la ejecución de la carga de trabajo. La información es utilizada para estimar los parámetros por nodo y posteriormente por el sistema completo. Para estimar la información de cada nodo basta con considerar la información de cada proceso que se ejecutó en dicho nodo, esto se logra sumando los valores obtenidos por proceso, después es necesario tomar en cuenta la descripción de cada nodo de la misma manera, y así obtener la evaluación del sistema completo.

Al aplicar el modelo de ejecución de máquina a los resultados obtenidos de los distintos contadores se modela el desempeño promedio del sistema, y tomando en cuenta el método de intersección de puntos, se expresa como un vector, tal vector es denominado vector de desempeño. Los valores contenidos dentro del vector representan la frecuencia con respecto a los ciclos de reloj en promedio necesarios para ejecutar la instrucción dada.

A continuación se presentan las gráficas de algunos de los resultados obtenidos al ejecutar las cargas de trabajo previamente descritas.

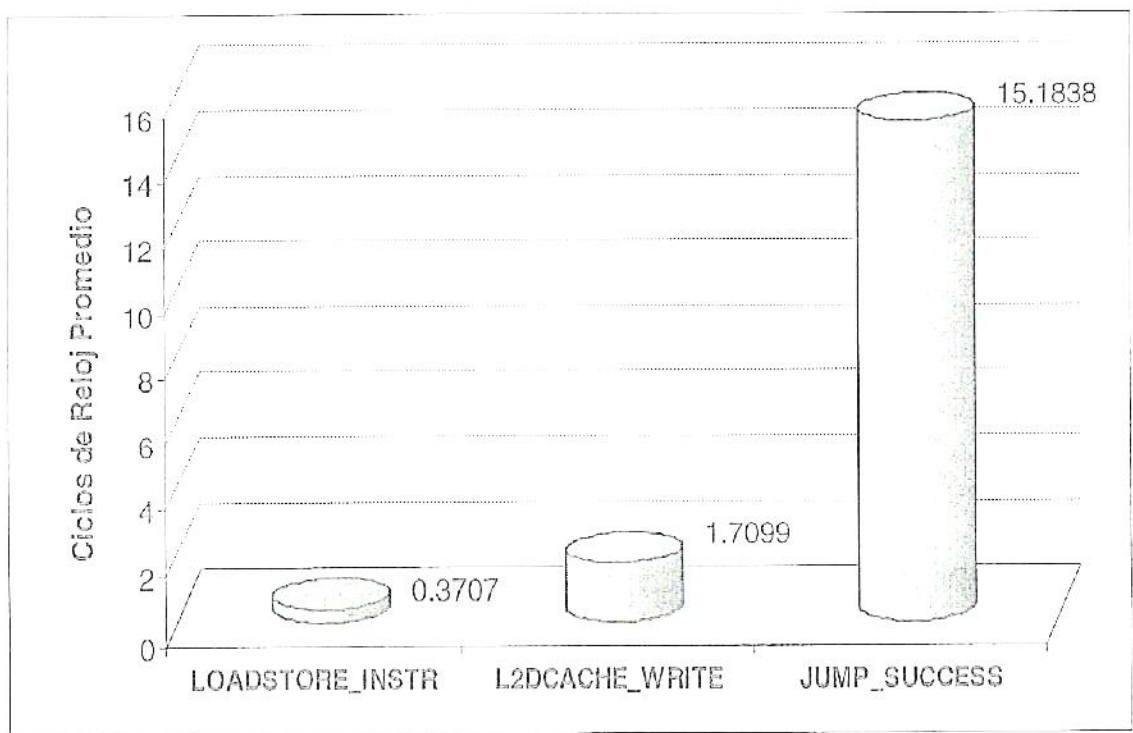


Figura 4.5: Vector de desempeño utilizando LOADSTORE_INSTR, L2DCACHE_WRITE, JUMP_SUCCES.

La figura 4.5, nos muestra el vector de desempeño generado por la ejecución de las 6 cargas de trabajo seleccionas y con la lectura de los contadores de desempeño LOADSTORE_INSTR, L2DCACHE_WRITE, JUMP_SUCCES. Los valores obtenidos para cada uno de los registros leídos son: 0.37, 1.7099 y 15.1838.

La figura 4.6 describe el vector compuesto por: L2CACHE_READWRITE, ELAPSED_CYCLES y JUMP. Con los valores 0.6933, 0.3091 y 0.1993.

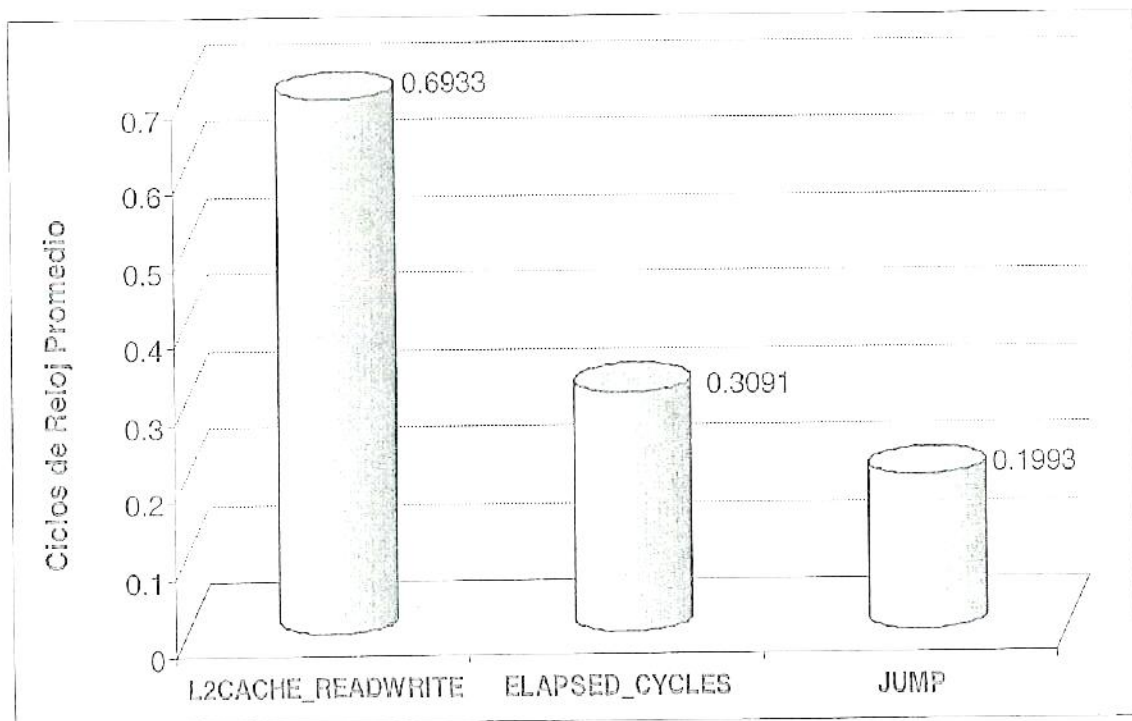


Figura 4.6: Vector de desempeño utilizando L2CACHE_READWRITE, ELAPSED_CYCLES y JUMP

La gráfica contenida en la figura 4.7, muestra el vector con: FP_INSTR, L2CACHE_READ y L2ICACHE_READ. Presentando los siguientes valores: 26.079, 361.4372, 506.5785.

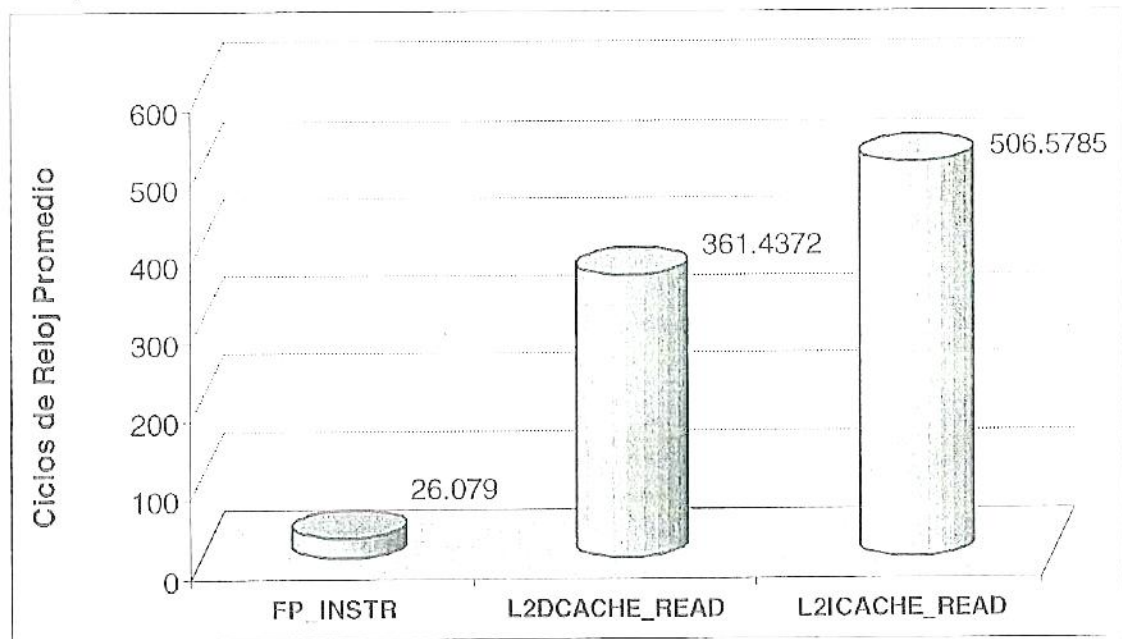


Figura 4.7: Vector de desempeño utilizando FP_INSTR, L2CACHE_READ y L2ICACHE_READ.

En la figura 4.8 se aprecia los datos L2CACHE_READWRITE, ELAPSED_CYCLES y JUMP. Con los valores obtenidos 0.6933, 0.3091 y 0.1993.

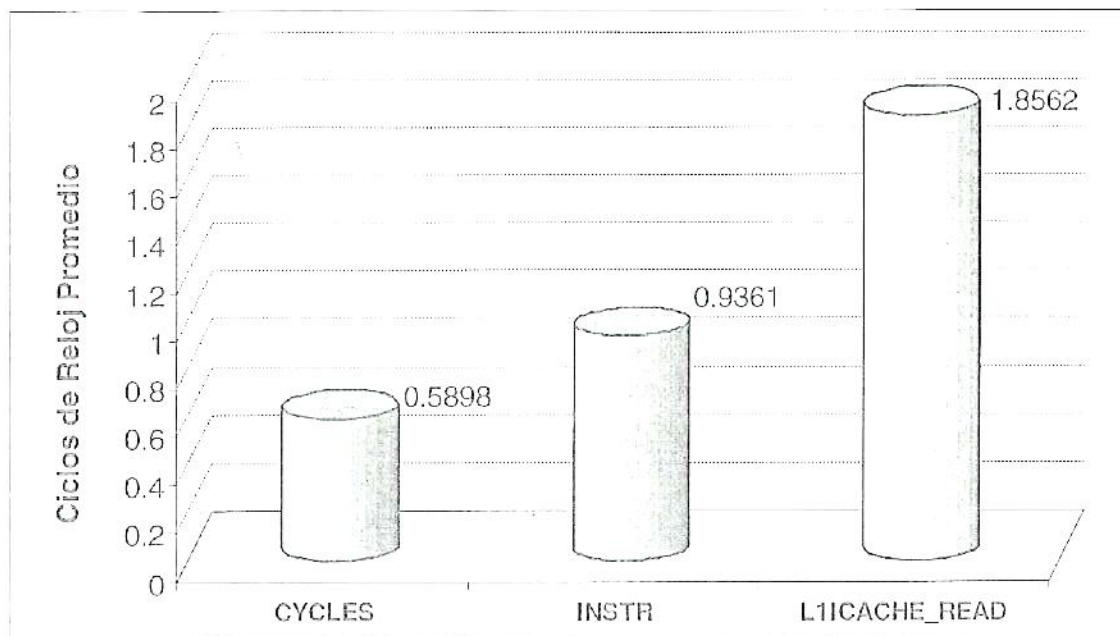


Figura 4.8: Vector de desempeño utilizando L2CACHE_READWRITE, ELAPSED_CYCLES y JUMP.

En conclusión se puede observar que con respecto a las operaciones de punto flotante, ciclos concluidos, saltos de procesador e instrucciones, el sistema opera de manera aceptable, debido a que algunos de los valores se encuentran en un rango creíble, los valores pueden no ser aceptados en cuanto a la dependencia lineal de la carga de las cargas de trabajo.

V Conclusiones

Introducción.

En este capítulo se presentan las conclusiones de la presente investigación, se toman en cuenta los objetivos generales y específicos, así como la delimitación del problema.

- Se accedió a los contadores de desempeño del microprocesador.
- Se modificó el código fuente de la biblioteca de paso de mensajes (*Mpich*), para obtener los valores de los contadores durante la ejecución de las cargas de trabajo.
- Se implementó la interfase gráfica de la herramienta *PEVTO* que permite generar de manera automática, cargas de trabajo paralelas y secuenciales.
- Se automatizó el proceso de selección de contadores de evaluación de rendimiento.
- Las cargas de trabajo generadas a partir del sistema *PEVTO*, arrojan los datos referentes a los contadores previamente seleccionados.
- La herramienta *PEVTO*, permite la evaluación de rendimiento a nivel proceso, nodo y sistema.

V.1 Trabajo futuro.

Se requiere implementar al sistema *PEVTO* con la capacidad de ser tolerante a fallas al momento de ejecución.

Es necesario implementar el algoritmo del punto medio operacional dentro de la interfase de la herramienta, de manera que la caracterización de los valores de los contadores pueda hacerse de manera automática.

La herramienta debe de tener la posibilidad de generar la configuración lógica de las partes del sistema distribuido que participarán en el experimento. Es decir, debe de dar la posibilidad de elegir en que nodos del sistema se ejecutará la carga de trabajo.

ANEXOS

Anexo A, Manual de usuario.

En este apéndice se describen las características del sistema, así como la manera en la que debe de ser operado. Tomando en cuenta de que todo lo necesario para que el sistema funcione ha sido previamente instalado en el sistema y este se encuentre listo.

El manual se encuentra estructurado en tres partes, ventana principal, sección de cargas distribuidas y la sección de cargas no distribuidas.

Ventana principal.



Figura 'Ventana Principal': Figura de la ventana principal de *PEVTO*.

Cuenta con un menú principal el cual tiene dos opciones '*Experimentos*' y

'Ayuda', en el menú 'Experimentos' se encuentran las opciones 'Distribuidos', 'No distribuidos' y 'Salir' (figura 'Menú Principal'). 'Distribuidos', da la lista de aplicaciones distribuidas disponible para perfilar, las aplicaciones paralelas se encuentran en un directorio dentro de la estructura de *PEVTO* llamado 'cargas_dist', las cargas se presentan en un cuadro de lista en otra ventana. 'No distribuidos', proporciona la lista de las aplicaciones que se pueden utilizar para generar códigos ejecutables capaces de obtener información del sistema que será evaluado, la lista es mostrada en una nueva ventana, los archivos son tomados del directorio 'cargas_no_dist'. La última opción de menú principal es 'Salir', la cual permite terminar con la operación de la aplicación.



Figura 'Menú Principal': Figura del menú principal de la aplicación.

El último menú de la ventana principal es el menú 'Ayuda', el cual tiene las opciones 'Manual' y 'Acerca de...'. 'Manual' presenta un texto con la descripción

del funcionamiento de la herramienta. 'Acerca de...' da un dialogo con los datos del autor y la licencia de distribución (figura 'Menú Ayuda').



Figura 'Menú Ayuda': Menú de ayuda de la ventana principal.

Ventana de cargas distribuidas.

En la figura 'Cargas distribuidas', se tiene la lista de las cargas paralelas disponibles para ser perfiladas, estos programas se encuentran en un directorio llamado '*cargas_dist*' en la estructura de *PEVTO*, en la ventana se tienen las opciones '*Agregar*', '*Siguiente ->*' y '*Cerrar*'. '*Agregar*' abre un dialogo de selección de archivos y al seleccionar un código y dar la opción de '*OK*', este código es colocado dentro del directorio de las cargas para la siguiente vez que se desee utilizar. '*Siguiente*' lleva a la ventana de selección de rutas de bibliotecas y '*Cerrar*' termina con la ejecución de esta ventana.

Ventana de selección de programas paralelos.

La ventana de selección de programas paralelos es un dialogo de archivo que al confirmar la selección agrega el archivo seleccionado al directorio de cargas distribuidas (figura 'Selección de cargas distribuidas').



Figura 'Cargas distribuidas': Ventana con los programas paralelos disponibles.

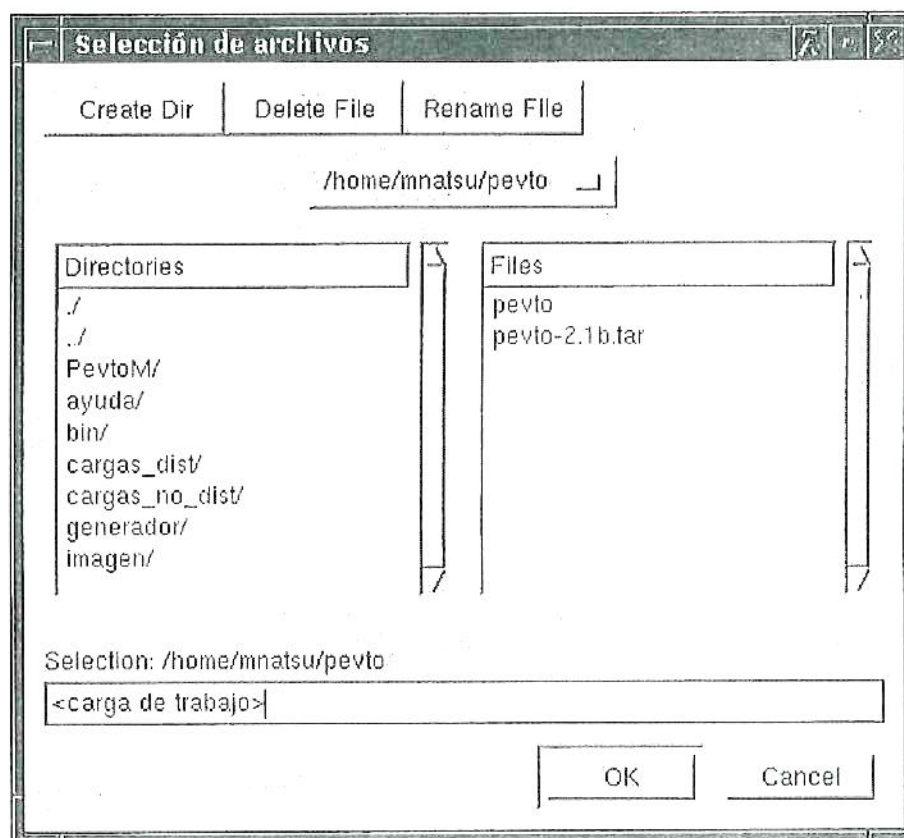


Figura 'Selección de cargas distribuidas': Ventana de selección de cargas.

Ventana de selección de contadores.

Los contadores se presentan en una ventana que tiene un cuadro de lista, la selección hecha es registrada en un archivo temporal, en la lista se muestran todas las posibles combinaciones de los contadores soportados por la plataforma Intel (figura 'Selección de contadores'), al seleccionar el par de contadores deseados basta con dar la opción '*Siguiente ->*' para continuar.

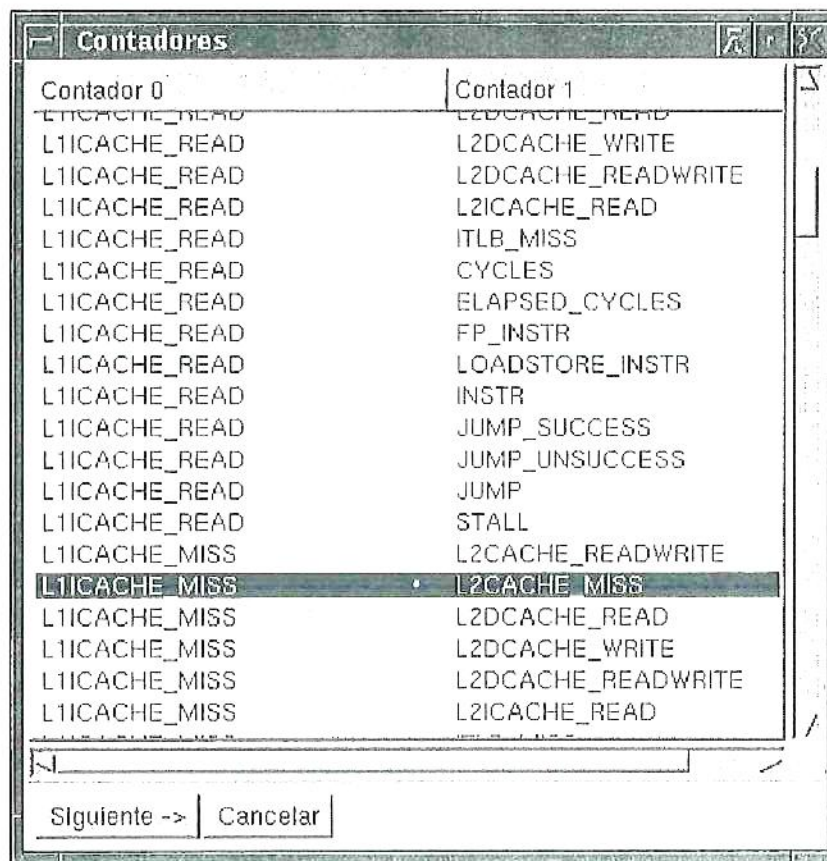


Figura 'Selección de contadores': Ventana que muestra los posibles contadores.

Ventana de selección de rutas.

Aquí se muestran rutas por omisión de las bibliotecas necesarias para generar código ejecutable de la carga seleccionada, así como las banderas necesarias para la fase de enlace en el proceso de compilación. Se pueden modificar estas rutas en cualquier momento antes de dar la opción de generar, solo es necesario con editar los campos de texto (figura 'Ruta de bibliotecas paralelas'). Para generar el ejecutable solo es necesario accionar la orden de

'Generar', presentada en el botón del mismo nombre.

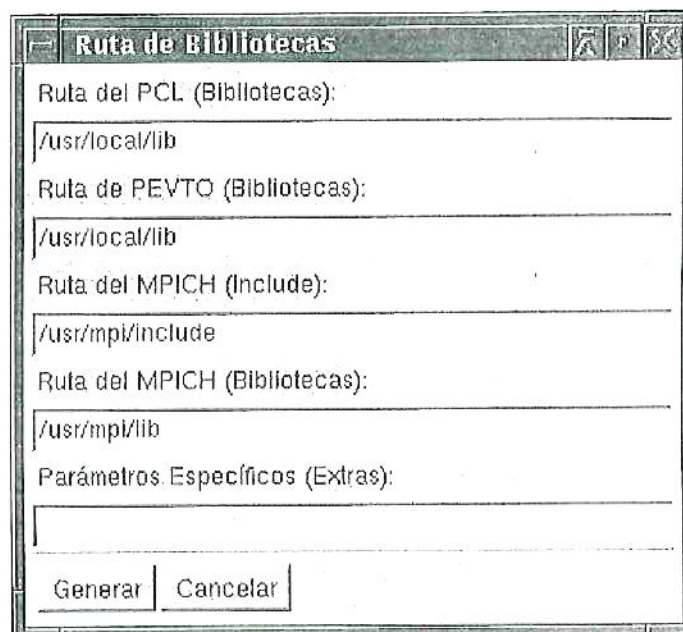


Figura 'Ruta de bibliotecas paralelas': Ventana de rutas para la fase de enlace.

Ventana de selección de programas no paralelos.

La figura 'Cargas no distribuidas', muestra la lista de programas no paralelos disponibles para perfilar. La ventana contiene 3 opciones 'Agregar', 'Siguiente ->' y 'Cerrar' (figura 'Cargas no distribuidas').

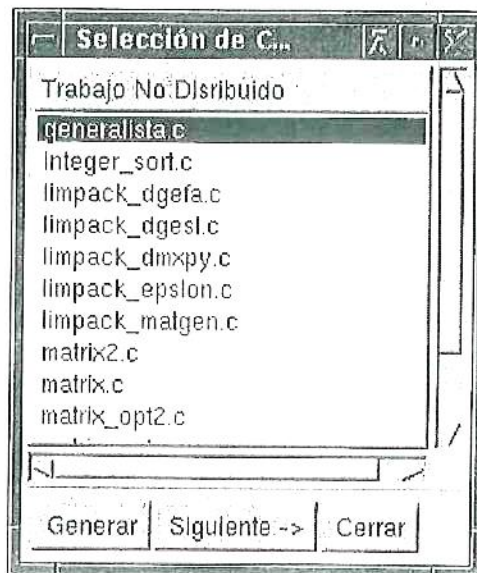


Figura 'Cargas no distribuidas': Ventana con los programas no paralelos disponibles.

'Agregar' permite incluir una carga no paralela a la lista de programas disponibles para generar el ejecutable perfilado. Se presenta un dialogo de selección de archivos (figura 'Selección de cargas no distribuidas'), y después de seleccionado el programa se confirma con la opción 'OK', los programas no paralelos que se desean incluir deben de estar escritos en un formato especial, el formato consta de un archivo no binario el cual contenga la carga de trabajo escrita en forma de función del lenguaje C. Es decir:

```
<vacío> nombre_de_la función(<vacío>)
{
    ...
    <carga>
    ...
}
```

Ejemplo:

```
void hola_mundo()
{
    int i,a,b,c;
    a=1;
    b=2;
    c=a+b;
    i=20;
    printf("Hola mundo %d %d \n",c,i);
}
```

Se genera un código en lenguaje *C* el cual tiene como nombre el nombre del archivo no binario con la función al mismo nombre con extensión *<nombre>.c*, el fuente generado tiene el siguiente formato:

```
#include <.....>
....
<función de perfilado>
<función de perfilado>
<vacío> nombre_de_la_función(<vacío>)
{
    ...
    <carga>
    ...
}
int main(argc,argv)
{
    <variables>
    <inicialización del PCL>
    <ciclo de eventos>
        <prueba del evento>
        <comienzo del evento (lectura del contador)>
        nombre_de_la_función(<vacío>)
        <detenemos el conteo>
        <imprimimos resultados>
    <fin del ciclo de eventos>
```

```
<Salida del PCL>  
}
```

La siguiente figura muestra el dialogo que se utiliza para incluir los archivos no distribuidos a la lista de disponibles.

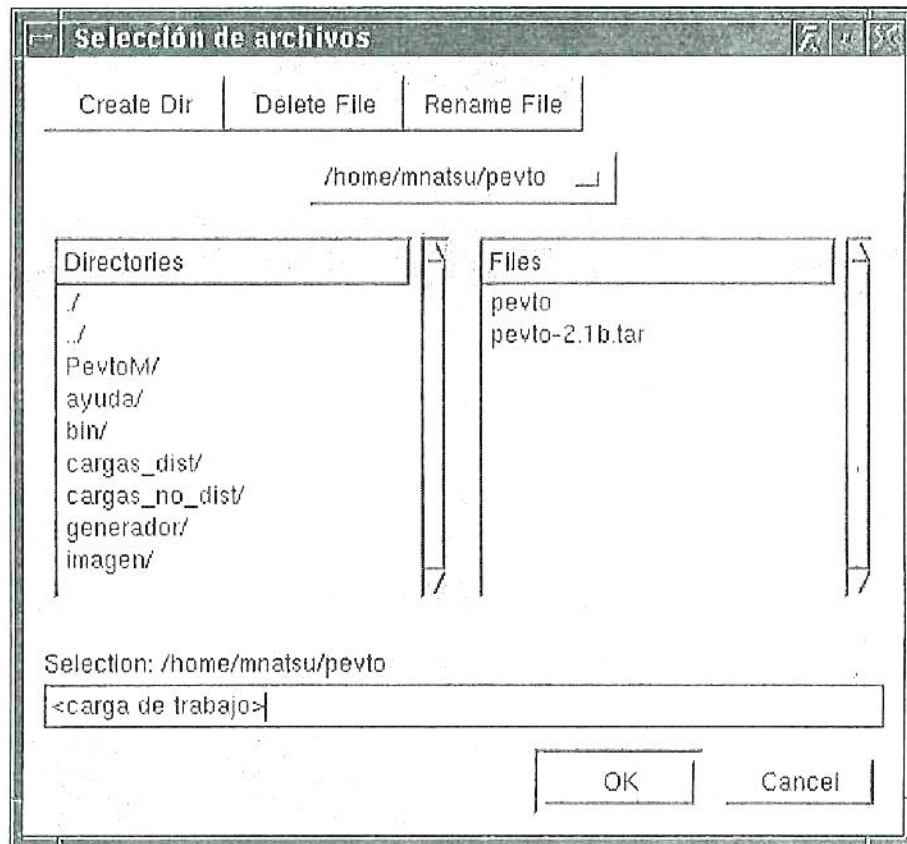


Figura 'Selección de cargas no distribuidas': Ventana de selección de cargas.

En la ventana de selección de cargas o paralelas, la opción 'Siguiente ->', lleva a seleccionar las rutas necesarias para generar los ejecutables perfilados capaces de leer los contadores del microprocesador. Por omisión se dan unas

rutas predefinidas, los cual puede ser adaptado a las necesidades de cada sistema al editar los campos de texto como se muestra en la figura 'Rutas de bibliotecas no paralelas'. Para obtener el código ejecutable perfilado es necesario accionar el botón con la etiqueta '*Generar*'.

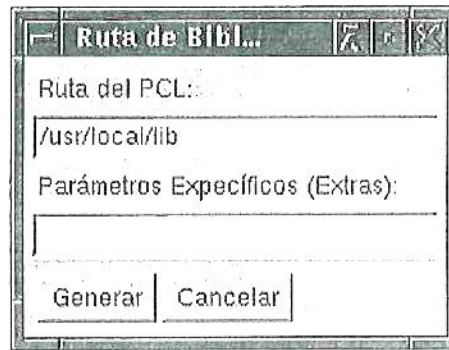


Figura 'Rutas de bibliotecas no paralelas': ventana de rutas para la fase de enlace.

Anexo B, Wrapper.

Generación automática de bibliotecas perfiladas.

Wrappergen

El generador de sobrecubiertas de perfilado (*wrappergen*) a sido diseñado para complementar la interfase de perfilado del *MPI*. Esto permite al usuario escribir cualquier número de *meta wrappers* los cuales pueden ser aplicados a cualquier cantidad de funciones de *MPI*. Los *Wrappers* pueden ser separados en archivos, y se pueden mezclar correctamente, así que mas de una capa de perfilado puede existir en una función individual.

Wrappergen necesita tres fuentes como entradas:

1. Una lista de funciones para las cuales se generaran las sobrecubiertas.
2. Declaración para las funciones que serán perfiladas. Para la velocidad y simplicidad en lo sintáctico, un formato especial se a utilizado. Ver el archivo *proto*. Las funciones de *MPI_1* se encuentran en *mpi_proto*. Las funciones de Entrada/Salida del *MPI-2* se encuentran en *mpiio_proto*.
3. Definiciones de sobrecubierta.

tiempos de salida. Que pueden ser escritos de igual manera al agregar el *MPI_Finalize* y *MPI_Init* a *mpiio_proto* o a través de un simple edit de la versión producida cuando se utiliza *mpi_proto* en lugar de *mpiio_proto*.

Escribiendo definiciones de sobrecubierta.

Las definiciones de sobrecubierta por si mismas consisten en código de C con *macros* especiales. Cada macros esta rodeado por `{{ }}`. Los siguientes macros están reconocidos por *wrappergen*:

`{{fileno}}`

Un índice integral representa de que archivo de sobrecubierta viene el macro. Esto es útil cuando se declaran variables globales para evitar colisiones en los nombres. Es recomendado que todo identificador declarado fuera de las funciones termine con `{{fileno}}`. Por ejemplo:

```
static double overhead_time_{{fileno}};
```

se expandirá a:

```
static double overhead_time_0;
```

tiempos de salida. Que pueden ser escritos de igual manera al agregar el *MPI_Finalize* y *MPI_Init* a *mpiio_proto* o a través de un simple edit de la versión producida cuando se utiliza *mpi_proto* en lugar de *mpiio_proto*.

Escribiendo definiciones de sobrecubierta.

Las definiciones de sobrecubierta por si mismas consisten en código de C con *macros* especiales. Cada macros esta rodeado por `{{ }}`. Los siguientes macros están reconocidos por *wrappergen*:

`{{fileno}}`

Un índice integral representa de que archivo de sobrecubierta viene el macro. Esto es útil cuando se declaran variables globales para evitar colisiones en los nombres. Es recomendado que todo identificador declarado fuera de las funciones termine con `{{fileno}}`. Por ejemplo:

```
static double overhead_time_{{fileno}};
```

se expandirá a:

```
static double overhead_time_0;
```

{{foreachfn}} es lo mismo que **{{forallfn}}** excepto que las sobrecubiertas son escritas solo para las funciones que se encuentran listadas explícitamente. Por ejemplo:

```
{{forallfn    fn_name    mpi_send    mpi_recv}}    static    int
{{fn_name}}_ncalls_{{fileno}};
{{endforallfn}}
```

se expandirá a:

```
static int MPI_Send_ncalls_2;

static int MPI_Recv_ncalls_2;
```

```
{{fnall <function name escape> <function A> <function B> ... }}
...
{{callfn}}
...
{{endfnall}}
```

{{fnall}} define una sobrecubierta que será usada en todas las funciones excepto en las funciones listadas. *Wrappergen* expandirá a definiciones

completas de funciones en formato tradicional de C. El macro `{{callfn}}` le indica al *wrappergen* donde insertar la llamada a la función que esta siendo perfilada. Debe de haber exactamente una instancia de la llamada al *macro* `{{callfn}}` en cada definición del *wrapper*. La sentencia especificada por `<function name escape>` se reemplazara por el nombre de cada función. Con una definición de *wrapper*, *macros* extra son reconocidos.

`{{vardecl <type> <arg> <arg> ... }}`

Se usa *vardecl* para declarar variables con definiciones de sobrecubierta. Si los macros mezclados llaman a variables a través de *vardecl* con los mismos nombres, *wrappergen* creará un único nombre adhiriendo números consecutivos al final del nombre requerido (*var*, *var1*, *var2*, ...) hasta que un nombre único es creado. No es recomendado declara variables de manera manual en una definición de *wrapper*, puede que los nombres de las variables colisionen con otros *wrappers*.

`{{<varname>}}`

Si una variable es declarada a través de *vardecl*, el nombre requerido para esa variable (que puede ser diferente del que aparecerá en el código final) se convierte en un macro temporal que se transformará en uno único. Por ejemplo.

{{vardecl int i d}}

se expandirá a:

`int i, d3;`

{{<argname>}}

Sugerido pero no necesario, un macro que consiste en el nombre de un argumento de la función que esta siendo perfilada se expandirá al nombre correspondiente al argumento. Esta opción de macro tiene el propósito de asegurar que la función que esta siendo perfilada tenga un argumento de un nombre dado.

{{<argnum>}}

Los argumentos de la función que está siendo perfilada también se debe de hacer referencia a ella por número, comenzando con el 0 y en incremento.

{{returnVal}}

returnVal expande a la variable usada para mantener el valor de la función que esta siendo perfilada.

{{callfn}}

callfn se expandirá a la función que está siendo perfilada. Con definiciones de wrapper anidadas, esto también representa el punto en donde insertar el código para cualquier función externa anidada. El orden de anidación es determinado por el orden en como encuentra el wrappergen a las definiciones de sobre cubierta. Por ejemplo, Si dos archivos *prof1.w* y *prof2.w* cada una con una definición para *MPI Send*, el código perfilado producido cuando se usan los archivos será de la siguiente forma:

```
int MPI_Send( args...)
arg declarations...
{
    /*pre-callfn code from wrapper 1 from prof1.w */
    /*pre-callfn code from wrapper 2 from prof1.w */
    /*pre-callfn code from wrapper 1 from prof2.w */
    /*pre-callfn code from wrapper 2 from prof2.w */

    returnVal = MPI_Send( args... );

    /*post-callfn code from wrapper 2 from prof2.w */
    /*post-callfn code from wrapper 1 from prof2.w */
    /*post-callfn code from wrapper 2 from prof1.w */
    /*post-callfn code from wrapper 1 from prof1.w */
    return returnVal;
}
```

{{fn <function name escape> <function A> <function B> ... }}

...

{{callfn}}

...

`{{endfnall}}`

fn es idéntico a *fnall* excepto que solo genera *wrappers* para las funciones nombradas explícitamente. Por Ejemplo:

```
{{fn this_fn MPI_Send}}  
  
{{vardecl int i}}  
  
{{callfn}}  
  
printf( "Call to {{this_fn}}.\n" );  
  
printf( "{{i}} was not used.\n" );  
  
printf( "The first argument to {{this_fn}} is {{0}}\n" );  
  
{{endfn}}
```

se expandirá a:

```
int MPI_Send( buf, count, datatype, dest, tag, comm )  
void * buf;  
int count;  
MPI_Datatype datatype;  
int dest;  
int tag; MPI_Comm comm;  
{  
    int returnVal;  
    int i;  
    returnVal = PMPI_Send( buf, count, datatype, dest, tag, comm );  
}
```

```
    printf( "Call to MPI_Send.\n" );  
    printf( "i was not used.\n" );  
    printf( "The first argument to MPI_Send is buf\n" );  
    return returnVal;  
}
```

{{fn_num}}

Este es un número, empezando de cero y se incrementa cada vez que se utiliza.

Anexo C, PCL.

Plataforma Intel Pentium Pro, Pentium II, Pentium III, Pentium IV.

Para mantener la compatibilidad binaria con los procesadores predecesores, el Pentium Pro, Pentium II y Pentium III tiene ocho registros de 32 bits de capacidad cada uno. El primer nivel del cache es de 8KB para las instrucciones (ICache) y 8KB para los datos (DCache) en la Pentium Pro y 16KB en ambos caches en la Pentium II y Pentium III. Como en la Pentium Pro, la Pentium II y Pentium III son microprocesadores CISC (complex instruction set computer), cada instrucción es dividida en micro operaciones (UOP's) de longitud fija. Dependiendo de la complejidad de la instrucción, está es dividida de 1-4 UOP's [Performance Counter Library, <http://www.intel.com>, 2002].

El Pentium Pro, Pentium II y Pentium III tiene dos contadores de desempeño capaces de contar un total de 77 eventos diferentes la mayoría se pueden leer dos al mismo tiempo aun que en PCL solo se pueden contar 22 de estos contadores de eventos, algunos de ellos con una mascara unitaria como parámetro para en el futuro dividir el tipo de evento. Algunos de los eventos solo pueden ser contados por un contador en específico. El Pentium III tiene 4

eventos adicionales concernientes al flujo de extensiones SMID. Los registros de los contadores son de 40 bits de ancho, el contador de la muestra de tiempo es de 64 bits de ancho. Con instrucciones especiales es posible escribir valores a los contadores de los registro de desempeño (*WRMSR*).

El contador de la muestra de tiempo, cuenta los ciclos de máquina transcurridos.

Ejemplo:

Ppro TSC: Ciclos de máquina transcurridos.

Los eventos contables por ambos contadores son:

Ppro 0 (43H DATA_MEM_REFS): Todas las referencias de memoria, ambas de cache y no de cache.

Ppro 1 (45H DCU_LINES_IN): Número de lineas alojadas en el primer nivel del cache de datos.

Ppro 2 (46H DCU_M_LINES_IN): Número de lineas alojadas en el primer nivel del cache y que tiene su estado modificado.

Ppro 3 (47H DCU_M_LINES_OUT): Número de lineas desubicadas en el primer nivel del cache de datos, que se encuentran marcadas como modificadas.

PPro 4 (48H DCU_MISS_OUTSTANDING): Número de ciclos medidos mientras el primer nivel de los fallos de cache esta pendiente. Un acceso que también fallo en L2 es cambiado por 2 ciclos. (e.j. Si se contaron N ciclos, deberán ser N+2 ciclos.) Cargas subsecuentes al mismo cache no resultaran en conteos adicionales. El conteo de los valores no es preciso, pero es útil.

PPro 5 (80H IFU_IFETCH): Número de instrucciones cargadas al primer nivel de cache.

PPro 6 (81H IFU_IFETCH_MISS): Número de fallos al primer nivel de cache.

PPro 7 (85H ITLB_MISS): Número de fallos al TLB (transfer look-aside buffer).

PPro 8 (86H IFU_MEM_STALL): Número de ciclos en los que la instrucción busca cuando el estado de la pipa se encuentra estancado.

PPro 9 (87H ILD_STALL): Número de ciclos donde el decodificador de longitud de instrucciones esta estancado.

PPro 10 (28H L2_IFETCH): Número de instrucciones buscadas del segundo nivel del cache.

PPro 11 (29H L2 LD): Número de datos cargados del segundo nivel del cache.

PPro 12 (2AH L2 ST): Número de datos guardados en el segundo nivel de cache.

PPro 13 (24H L2 LINES IN): Número de líneas alojadas en el segundo nivel de cache.

PPro 14 (26H L2 LINES OUT): Número de líneas de cache removidas del segundo nivel de cache.

PPro 15 (25H L2 M LINES INM): Número de líneas alojadas en el segundo nivel del cache que han sido modificadas.

PPro 16 (27H L2 M LINES OUTM): Número de líneas modificadas en el segundo nivel de cache que han sido removidas.

PPro 17 (2EH L2 RQSTS): Número de peticiones al segundo nivel de cache.

PPro 18 (21H L2 ADS): Número de direcciones del segundo nivel de cache del bus paradas.

PPro 19 (22H L2 DBUS BUSY): Número de ciclos durante los cuales el bus de datos estuvo ocupado.

PPro 20 (23H L2 DBUS BUSY RD): Número de ciclos durante los cuales el bus de datos está ocupado transfiriendo datos del segundo nivel del cache al procesador.

PPro 21 (62H BUS DRDY CLOCKS): Número de ciclos que la señal DRDY estuvo activa.

PPro 22 (63H BUS LOCK CLOCKS): Número de ciclos de reloj durante los cuales la señal LOCK es afirmada.

PPro 23 (60H BUS REQ OUTSTANDING): Número de peticiones de bus pendientes que resultan de una petición de lectura de las líneas en el primer nivel de cache de datos.

PPro 24 (65H BUS TRAN BRD): Número de transacciones de lectura fallaron o se rompieron.

PPro 25 (66H BUS TRAN RFO): Número de lecturas hechas por el dueño de la transacción.

PPro 26 (67H BUS TRANS WB): Número de respuestas de las transacciones.

PPro 27 (68H BUS TRAN IFETCH): Número de instrucciones de búsqueda de transacciones.

PPro 28 (69H BUS TRAN INVALID): Número de transacciones de invalidación de bus completadas.

PPro 29 (6AH BUS TRAN PWR): Número de transacciones de escritura parcial completada.

PPro 30 (6BH BUS TRANS P): Número de transacciones parciales completadas.

PPro 31 (6CH BUS TRANS IO): Número transacciones de I/O completadas.

PPro 32 (6DH BUS TRAN DEF): Número de transacciones aplazadas completadas.

PPro 33 (6EH BUS TRAN BURST): Número de transacciones de *bus* completadas.

PPro 34 (70H BUS TRAN ANY): Número de todas las transacciones

completadas.

PPro 35 (6FH BUS TRAN MEM): Número de transacciones de memoria completadas.

PPro 36 (64H BUS DATA RCV): Número de ciclos de reloj del *bus* durante los cuales el procesador recibe datos.

PPro 37 (61H BUS BNR DRV): Número de ciclos de reloj de *bus* durante los cuales el procesador esta manejando el pin BNR.

PPro 38 (7AH BUS HIT DRV): Número de ciclos de reloj de *bus* durante los cuales el procesador está manejando el pin HIT incluyendo los ciclos convenientes de observación de la casilla.

PPro 39 (7BH BUS HITM DRV): Número de ciclos de reloj de *bus* durante los cuales el procesador está manejando el pin HITM incluyendo los ciclos convenientes de observación de la casilla.

PPro 40 (7EH BUS SNOOP STALL): Número de ciclos de reloj durante los cuales la vigilancia del *bus* se encuentra parada.

PPro 41 (03H LD BLOCKS): Número de seguros almacenados del *buffer*.

PPro 42 (04H SB DRAINS): Número de ciclos en los cuales el almacenamiento del buffer esta bloqueado.

PPro 43 (05H MISALIGN MEM REF): Número de referencia de memoria desalineadas.

PPro 44 (C0H INST RETIRED): Número de instrucciones retiradas.

PPro 45 (C2H UOPS RETIRED): Número de micro-operaciones retiradas.

PPro 46 (D0H INST DECODER): Número de instrucciones decodificadas y traducidas a *UOP's*.

PPro 47 (C8H HW INT RX): Número de interrupciones de hardware recibidas.

PPro 48 (C6H CYCLES INT MASKED): Número de ciclos de procesador para los cuales las interrupciones están deshabilitadas.

PPro 49 (C7H CYCLES INT PENDIND AND MASKED): Número de ciclos de procesador para los cuales las interrupciones están deshabilitadas y pendientes.

PPro 50 (C4H BR INST RETIRED): Número de ramas de instrucciones retiradas.

PPro 51 (C5H BR MISS PRED RETIRED): Número de ramas completadas pero malprededidas.

PPro 52 (C9H BR TAKEN RETIRED): Número de ramas tomadas completadas.

PPro 53 (CAH BR MISS PRED TAKEN RET): Número de ramas tomadas completadas, pero malprededidas.

PPro 54 (E0H BR INST DECODED): Número de instrucciones de ramas decodificadas.

PPro 55 (E2H BTB MISSES): Número de ramas que fallaron el *BTB*.

PPro 56 (E4H BR BOGUS): Número de ramas falsas.

PPro 57 (E6H BACLEARs): Número de veces que el BACLEAR-signal es acertado.

PPro 58 (A2H RESOURCE STALLS): Número de ciclos durante los cuales hay

recursos relacionados a los paros.

PPro 59 (D2H PARTIAL RAT STALLS): Número de ciclos o eventos para paros parciales.

PPro 60 (06H SEGMENT REG LOADS): Número de segmentos de cargas registradas.

PPro 61 (79H CPU CLK UNHALTED): Número de ciclos durante los cuales el procesador no está parado.

PPro 62 (B0H MMX INSTR EXEC): Número de instrucciones MMX ejecutadas.

PPro 63 (B3H MMX INSTR TYPE EXEC): Número de instrucciones MMX ejecutadas. El otro parámetro es la unidad de mascara que especifica que categoría deberá de ser contada.

PPro 64 (B1H MMX SAT INSTR EXEC): Instrucciones MMX ejecutadas saturadas.

PPro 65 (B2H MMX uOPS EXEC): Número de UOPs MMX ejecutados.

PPro 66 (CCH FP MMX TRANS): Transiciones de instrucciones MMX a instrucciones FP.

PPro 67 (CDH MMX ASSIST): Número de asistentes MMX (instrucciones EMMS ejecutadas).

PPro 68 (CEH MMX INSTR RET): Número de instrucciones MMX retiradas.

PPro 69 (D4H SEG RENAME STALLS): Registros del segmento de paradas renombradas.

PPro 70 (D5H SEG REG RENAMES): Registros de segmento renombrados.

PPro 71 (D6H RET SEG RENAMES): Número de registros del segmento renombrados de eventos retirados.

PPro 72 (D8H EMON SSE INST RETIRED): Número de flujo SIMD de extensiones retiradas.

PPro 73 (D9H EMON SSE COMP INST RET): Número de extensiones de instrucciones de flujo SIMD retiradas.

PPro 74 (07H EMON SSE PRE DISPATCHED):. Número de instrucciones ordenadas enviadas preobtenidas/debilmente (inclusive preobtenidas especulativamente).

PPro 75 (4BH EMON SSE PRE MISS): Número de instrucciones preobtenidas/debilmente ordenadas que fallan en todos los caches.

Eventos exclusivos del contador:

- Específicamente del contador 0:

PPro C0_0 (C1H FLOPS): Número de instrucciones de punto flotante retiradas.

Pro C0_1 (10H FP COMP OPS EXE): Número de operaciones de punto flotante comenzadas (pero precisamente terminadas.)

PPro C0_2 (14H CYCLES DIV BUSY): Número de ciclos durante los cuales el divisor se encuentra ocupado.

- Específicamente del contador 1:

PP1 0 (11H FP ASSIST): Número de excepciones de punto flotante controladas por un microcódigo.

PP1 1 (12H MUL): Número de multiplicaciones (enteras y punto

flotantes).

PP1 2 (13H DIV): Número de divisiones (enteras y punto flotantes).

Todos los eventos que existen en el Pentium Pro también existen en el Pentium II y Pentium III. El Pentium II y Pentium III tienen eventos adicionales definidos principalmente para las extensiones *MMX*.

Dentro de *PCL* los nombres de los contadores difieren un poco, esto con la finalidad de tener una lista en común para diferentes arquitecturas. Es decir, *PCL* maneja una sola lista para un grupo de procesadores de distintos fabricantes. A continuación se muestra la lista de eventos que proporciona *PCL* para todas sus arquitecturas soportadas.

TABLA 1. Eventos que pueden ser medidos en <i>PCL</i> .	
PCL_EVENT	Evento Medido
PCL_L1CACHE_READ	L1 (Nivel uno) lecturas de cache
PCL_L1CACHE_WRITE	L1 escrituras de cache
PCL_L1CACHE_READWRITE	L1 lecturas y escrituras de cache
PCL_L1CACHE_HIT	L1 aciertos al cache
PCL_L1CACHE_MISS	L1 fallos al cache

PCL_L1DCACHE_READ	L1 lecturas de datos de cache
PCL_L1DCACHE_WRITE	L1 escrituras de datos de cache
PCL_L1DCACHE_READWRITE	L1 escrituras y lecturas de datos de cache
PCL_L1DCACHE_HIT	L1 aciertos a los datos de cache
PCL_L1DCACHE_MISS	L1 fallos a los datos de cache
PCL_L1ICACHE_READ	L1 instrucciones de cache leídas
PCL_L1ICACHE_WRITE	L1 instrucciones de cache escritas
PCL_L1ICACHE_READWRITE	L1 instrucciones de cache leídas y escritas
PCL_L1ICACHE_HIT	L1 aciertos de instrucciones de cache
PCL_L1ICACHE_MISS	L1 fallos de instrucciones de cache
PCL_L2CACHE_READ	L2 (Nivel dos) lecturas de cache
PCL_L2CACHE_WRITE	L2 escrituras de cache
PCL_L2CACHE_READWRITE	L2 lecturas y escrituras de cache
PCL_L2CACHE_HIT	L2 aciertos a cache
PCL_L2CACHE_MISS	L2 fallos a cache
PCL_L2DCACHE_READ	L2 lecturas de datos de cache
PCL_L2DCACHE_WRITE	L2 escrituras de datos de cache
PCL_L2DCACHE_READWRITE	L2 escrituras y lecturas de datos de cache
PCL_L2DCACHE_HIT	L2 aciertos de datos de cache

PCL_L2DCACHE_MISS	L2 fallos de datos de cache
PCL_L2ICACHE_READ	L2 instrucciones de cache leídas
PCL_L2ICACHE_WRITE	L2 instrucciones de cache escritas
PCL_L2ICACHE_READWRITE	L2 instrucciones de cache leídas y escritas
PCL_L2ICACHE_HIT	L2 aciertos de instrucciones de cache
PCL_L2ICACHE_MISS	L2 fallos de instrucciones de cache
PCL_TLB_HIT	TLB (Translation Lookaside Buffer) aciertos
PCL_TLB_MISS	TLB fallos
PCL_ITLB_HIT	Aciertos de instrucciones de TLB
PCL_ITLB_MISS	Fallos de instrucciones de TLB
PCL_DTLB_HIT	Aciertos de datos de TLB
PCL_DTLB_MISS	Fallos de datos de TLB
PCL_CYCLES	Ciclos
PCL_ELAPSED_CYCLES	Ciclos transcurridos
PCL_INTEGER_INSTR	Instrucciones enteras ejecutadas
PCL_FP_INSTR	Instrucciones de punto flotante ejecutadas
PCL_LOAD_INSTR	Instrucciones cargadas y ejecutadas
PCL_STORE_INSTR	Instrucciones almacenadas y ejecutadas

PCL_LOADSTORE_INSTR	Instrucciones cargadas y almacenadas ejecutadas
PCL_INSTR	Instrucciones ejecutadas
PCL_JUMP_SUCCESS	Salto ejecutados concluidos
PCL_JUMP_UNSUCCESS	Salto ejecutados no concluidos
PCL_JUMP	Salto ejecutados
PCL_ATOMIC_SUCCESS	Instrucciones atómicas ejecutadas con éxito
PCL_ATOMIC_UNSUCCESS	Instrucciones atómicas ejecutadas sin éxito
PCL_ATOMIC	Instrucciones atómicas ejecutadas
PCL_STALL_INTEGER	Paros de enteros
PCL_STALL_FP	Paros de punto flotante
PCL_STALL_JUMP	Paro de saltos
PCL_STALL_LOAD	Paro de carga
PCL_STALL_STORE	Paro de almacén
PCL_STALL	Paros
PCL_MFLOPS	Millones de operaciones de punto flotante/segundo
PCL_IPC	Instrucciones ejecutadas por ciclo
PCL_L1DCACHE_MISSRATE	Level 1 data cache miss rate
PCL_L2DCACHE_MISSRATE	Level 2 data cache miss rate

PCL_MEM_FP_RATIO	Razón de accesos a memoria de las operaciones de punto flotante
------------------	---

Los 22 eventos legibles de Intel PIII e Intel PIV se encuentran remarcados con negritas.

Anexo D, Código de *PEVTO*.

Módulo principal.

Se determina de tipo de lenguaje en el que está escrito el programa.

```
#!/usr/bin/perl -w
```

Se incluye el módulo que permite el uso de Gtk.

```
use Gtk;
```

Se inicializa el Gtk.

```
set_locale Gtk;  
init Gtk;
```

Especificamos cuales son los módulos que se necesitan para el funcionamiento de las diferentes partes de *PEVTO*, estos módulos son parte de la aplicación. Es decir fueron programados junto con la aplicación.

```
require PevtoM::carga_no_distribuida;  
require PevtoM::carga_distribuida;  
require PevtoM::ayuda_manual;
```

Definimos las banderas de Falso y Verdadero.

```
my $false = 0;  
my $true = 1;
```

Damos un arreglo que contenga la estructuras de los menus de opción para nuestra interfase.

```
my @cosas_menu = ( { path => '/_Experimentos',
```

```

        type      => '<Branch>' },
    { path      => '/Experimentos/_Distribuidos',
      accelerator => '<control>D',
      callback    => \&carga_distribuida },
    { path      => '/Experimentos/_No Distribuidos',
      accelerator => '<control>N',
      callback    => \&carga_no_distribuida },
    { path      => '/Experimentos/sep1',
      type       => '<Separator>' },
    { path      => '/Experimentos/_Salir',
      accelerator => '<control>X',
      callback    => sub { Gtk->exit( 0 ); } },
    { path      => '/_Ayuda',
      type       => '<LastBranch>' },
    { path      => '/Ayuda/_Manual',
      accelerator => '<control>M',
      callback    => \&ayuda_manual},
    { path      => '/Ayuda/sep2',
      type       => '<Separator>'},
    { path      => '/Ayuda/_Acerca de...',
      accelerator => '<control>A',
      callback    => \&acerca_de} );

```

Declaración de las variables específicas de este módulo.

```

my $ventana;
my $principal_vbox;
my $barra_menu;
my $pixmap;
my $fixed;
my $gdcpixmap;
my $mask;
my $style;
my $normal;
my $gc;

```

Limpieza de los archivos temporales antes de empezar con cualquier acción.

```

system ("rm .tmp* 2>/dev/null");

```

Creación de la ventana principal de la aplicación.

```

# Creamos la ventana principal
$ventana = new Gtk::Window( "toplevel" );
$ventana->signal_connect( "destroy", sub { Gtk->exit( 0 ); } );
$ventana->set_title( "PEVTO" );
$ventana->set_usize( 300, 200 );
$ventana->show();

```

Creamos y agregamos un cuadro de empaquetado vertical a la ventana.

```
$principal_vbox = new Gtk::VBox( $false, 1 );
$principal_vbox->border_width( 1 );
$ventana->add( $principal_vbox );
$principal_vbox->show();
```

Creamos el menú donde se pondrán las opciones principales.

```
# Creamos el menú.
$barra_menu = crea_barra_menu( $ventana );
$principal_vbox->pack_start( $barra_menu, $false, $true, 0 );
$barra_menu->show();
```

Iniciamos y cargamos la imagen de fondo de la aplicación.

```
# Creamos el pixmap y los widgets necesarios.
$style = Gtk::Widget->get_default_style();
$normal = $style->bg( 'normal' );
$gc = $style->black_gc;
( $gdkpixmap, $mask ) = Gtk::Gdk::Pixmap->create_from_xpm(
$ventana->window, $normal, "imagen/logo.xpm" );
```

Creamos un contenedor de la imagen.

```
$pixmap = new Gtk::Pixmap( $gdkpixmap, $mask );
$pixmap->show();
```

Ponemos el elemento donde se desplegará la imagen, lo agregamos al empaquetado vertical y lo mostramos.

```
# Para desplegar el pixmap, creamos un widget fixed.
$fixed = new Gtk::Fixed();
$fixed->set_usize( 300, 175 );
$fixed->put( $pixmap, 0, 0 );
$principal_vbox->add( $fixed );
$fixed->show();
```

Mostramos la ventana principal e iniciamos el ciclo principal de ejecución.

```
$ventana->show();
$ventana->grab_add();
main Gtk;
exit( 0 );
```

1. Bloque de subrutinas del módulo principal PEVTO.

Función que crea la barra de menú, iniciamos los elementos del menú, y regresamos la barra de menú.

```
sub crea_barra_menu
{
    my ( $ventana ) = @_ ;

    my $barra_menu;
    my $item_factory;
    my $accel_group;

    $accel_group = new Gtk::AccelGroup();
    $item_factory = new
Gtk::ItemFactory( 'Gtk::MenuBar', '<main>', $accel_group );
    $item_factory->create_items( @cosas_menu );

    # Aderimos el grupo de aceleración..
    $ventana->add_accel_group( $accel_group );
    return ( $item_factory->get_widget( '<main>' ) );
}
```

Se crea el cuadro de dialogo para la ayuda, la etiqueta con el texto y se agrega y se pone listo para visualizar.

```
sub acerca_de
{
    my $dialogo= new Gtk::Dialog();
    $dialogo->set_title( "PEVTO Acerca de" );
    $dialogo->set_usize( 300, 250 );
    $dialogo->signal_connect("destroy",sub { $dialogo->destroy();});
    $dialogo->border_width(10);

    my $boton= new Gtk::Button( "Close" );
    $dialogo->action_area->pack_start( $boton, $false, $false, 0 );
    $boton->signal_connect("clicked",sub { $dialogo->destroy();});
    $boton->show();

    my $etiqueta = new Gtk::Label( "\nPEVTO\n\n" );
    $dialogo->vbox->pack_start( $etiqueta, $false, $false, 0 );
    $etiqueta->show();
}
```



```

    $etiqueta = new Gtk::Label( "PEVTO, permite perfialr el
desempeño\nde programas distribuidos y no distribuidos,\nPEVTO es
ditribuido bajo la licencia GPL.\n\nAutor: Masaru Natsu
Cárdenas\nEmail: mnatsu\@fciencias.ens.uabc.mx" );
    $dialogo->vbox->pack_start( $etiqueta, $false, $false, 0 );
    $etiqueta->show();

    $dialogo->show();
    $dialogo->grab_add();
}
# Fin del Programa.

```

Módulo de control de las cargas distribuidas.

En está sección se presenta el módulo que se encarga del control y generación de las cargas distribuidas instrumentadas.

La primera función abre el directorio donde se encuentran las cargas las lee y forma una lista con ellas que estarán disponibles para la selección del usuario.

```

sub genera_lista_dist
{
    my $clist=$_[0];
    my @lista;
    unless (opendir(CDIR,"cargas_dist"))
    {
        warn ("No se puede leer el actual
directorio\n");
    }
    my $i=0;
    foreach $archivo (readdir(CDIR))
    {
        if (($archivo ne ".") && ($archivo ne ".."))
        {
            $lista[0]=$archivo;
            $clist->append(@lista);
        }
    }
    closedir(CDIR);

    return;
}

```

```
}
```

La present función obtiene el valor seleccionado por e usuario de la lista presentada en la ventana de selección.

```
# Función que obtiene la información del cuadro de lista.
sub en_seleccion_dist
{
    my ( $clist, $renglon, $columna, $evento, @dato ) = @_;

    my $texto;

    # Obtenemos el texto almacenado en el renglon y columna
    # en el que se hizo click. Lo recibimos con un apuntador en el
    # argumento de texto.
    $texto = $clist->get_text( $renglon, $columna );

    # Solo imprimimos alguna información sobre el texto
    open (TMP,"> .tmp_selec_dist");
    print TMP "$texto";
    close( TMP );
    #print( "\t\t$texto\n\n" );

    return;
}
```

En está función se toman todos los datos recabados durante la ejecución de la aplicación y se genera un código ejecutable en base a las opciones.

```
# Función que genera el ejecutable.
sub genera_dist
{
    my ($widget, $entrada)=@_;
    my @librerias;
    my $i=0;
    my $linea;
    my $carga=0;
    my $lib=0;

    unless(open ( ARCH_LIB,"< .tmp_contadores_lib"))
    {
        warn "Advertencia: No se an seleccionado los contadores\n";
    }
}
```

```

else
{
    $lib=<ARCH_LIB>;
    close( ARCH_LIB );
}

open( ARCH_P1, "< .tmp_dist_lib1");
my $ruta1=<ARCH_P1>;
close( ARCH_P1 );

open( ARCH_P2, "< .tmp_dist_lib2");
my $ruta2=<ARCH_P2>;
close( ARCH_P2 );

open( ARCH_P3, "< .tmp_dist_lib3");
my $ruta3=<ARCH_P3>;
close( ARCH_P3 );

open( ARCH_P4, "< .tmp_dist_lib4");
my $ruta4=<ARCH_P4>;
close( ARCH_P4 );

open( ARCH_P5, "< .tmp_dist_extra");
my $ruta5=<ARCH_P5>;
close( ARCH_P5 );

unless(open(ARCH_C, "< .tmp_selec_dist"))
{
    warn "Advertencia: No se a seleccionado la carga\n";
}
else
{
    $carga=<ARCH_C>;
    close( ARCH_C );
}

open(LIB, "< generador/lista");
foreach $linea (<LIB>)
{
    chomp($linea);
    $arreglo[$i]=$linea;
    $i++;
}
close( LIB );
if ($carga && $lib)
{
    my $cadena="gcc cargas_dist/" . $carga . " -o bin/";
    chop($carga);
    chop($carga);
    $cadena=$cadena.$carga." -I".$ruta3." -L".$ruta2."
-1".$arreglo[$lib]." -L".$ruta1." -lpcl -L".$ruta4." -lpmpich"."

```

```

".$ruta5;
    system ($cadena);
}
else
{
    warn "Advertencia: No se a definido la Carga o los Contadores
para el experimento\n\n";
}
return;
}

```

Funciones que capturan la información de las entradas de las rutas de las bibliotecas que se le presentan al usuario, como la información es independiente. Las entradas son colocadas en archivos temporales que sirvan de referencia en otros módulos.

```

# Funciones para obtener las rutas

sub entrada_modificada1
{
    my ( $widget, $entrada ) = @_;

    my $entrada_texto = $entrada->get_text();
    open (TMP,"> .tmp_dist_lib1");
    print TMP "$entrada_texto";
    close( TMP );
    return;
}

sub entrada_modificada2
{
    my ( $widget, $entrada ) = @_;

    my $entrada_texto = $entrada->get_text();
    open (TMP,"> .tmp_dist_lib2");
    print TMP "$entrada_texto";
    close( TMP );
    return;
}

sub entrada_modificada3
{
    my ( $widget, $entrada ) = @_;

```

```

    my $entrada_texto = $entrada->get_text();
    open (TMP,"> .tmp_dist_Inc");
    print TMP "$entrada_texto";
    close( TMP );
    return;
}

sub entrada_modificada4
{
    my ( $widget, $entrada ) = @_;

    my $entrada_texto = $entrada->get_text();
    open (TMP,"> .tmp_dist_lib4");
    print TMP "$entrada_texto";
    close( TMP );
    return;
}

sub entrada_modificada5
{
    my ( $widget, $entrada ) = @_;

    my $entrada_texto = $entrada->get_text();
    open (TMP,"> .tmp_dist_extra");
    print TMP "$entrada_texto";
    close( TMP );
    return;
}

```

En esta función se le presenta al usuario la posibilidad de proporcionar las rutas hacia las diferentes bibliotecas que son necesarias para la generación de ejecutables capaces de perfilar el sistema. Así de la misma forma se puede apreciar en el código la relación de la interface con las funciones que controla la información proporcionada.

```

# Función para generar el ejecutable del paquete distribuido.
sub genera_binario
{
    my $ventana;
    my $vbox;
    my $hbox;
    my $hbox_e;
    my $entradal;
    my $entrada2;

```

```

my $entrada3;
my $entrada4;
my $entrada5;
my $boton;
my $boton_cancel;
my $etiqueta1;
my $etiqueta2;
my $etiqueta3;
my $etiqueta4;
my $separador;

# Create a window
$ventana = new Gtk::Window( "toplevel" );
$ventana->set_usize( 330, 281 );
$ventana->set_title( "Ruta de Bibliotecas" );
$ventana->border_width(1);
$ventana->signal_connect( "destroy", sub { $ventana->destroy();
} );

$ vbox = new Gtk::VBox( 0, 2 );
$ vbox->border_width(3);
$ ventana->add( $ vbox );
$ vbox->show();

$ hbox_e= new Gtk::HBox(0, 2);
$ hbox_e->border_width(3);
$ vbox->add($ hbox_e);
$ hbox_e->show();

$ etiqueta = new Gtk::Label( "Ruta del PCL (Bibliotecas):" );
$ etiqueta->set_justify( 'left' );
$ hbox_e->pack_start($ etiqueta,0,0,0);
$ etiqueta->show();

# Creamos la entrada de texto.
$ entrada1 = new Gtk::Entry( 255 );
$ entrada1->signal_connect( "changed", \&entrada_modificada1,
$ entrada1 );
$ entrada1->set_text( "/usr/local/lib" );
$ vbox->pack_start( $ entrada1, 0, 0, 0 );
$ entrada1->show();

$ hbox_e= new Gtk::HBox(0, 2);
$ hbox_e->border_width(3);
$ vbox->add($ hbox_e);
$ hbox_e->show();

$ etiqueta = new Gtk::Label( "Ruta de PEVIO (Bibliotecas):" );
$ etiqueta->set_justify( 'left' );
$ hbox_e->pack_start($ etiqueta,0,0,0);
$ etiqueta->show();

```



```

$entrada2 = new Gtk::Entry( 255 );
$entrada2->signal_connect( "changed", \&entrada_modificada2,
$entrada2 );
$entrada2->set_text( "/usr/local/lib" );
$vbbox->pack_start( $entrada2, 0, 0, 0 );
$entrada2->show();

$hbox_e= new Gtk::HBox(0, 2);
$hbox_e->border_width(3);
$vbbox->add($hbox_e);
$hbox_e->show();

$etiqueta = new Gtk::Label( "Ruta del MPICH (Include):" );
$etiqueta->set_justify( 'left' );
$hbox_e->pack_start($etiqueta,0,0,0);
$etiqueta->show();

$entrada3 = new Gtk::Entry( 255 );
$entrada3->signal_connect( "changed", \&entrada_modificada3,
$entrada3 );
$entrada3->set_text( "/usr/mpi/include" );
$vbbox->pack_start( $entrada3, 0, 0, 0 );
$entrada3->show();

$hbox_e= new Gtk::HBox(0, 2);
$hbox_e->border_width(3);
$vbbox->add($hbox_e);
$hbox_e->show();

$etiqueta = new Gtk::Label( "Ruta del MPICH (Bibliotecas):" );
$etiqueta->set_justify( 'left' );
$hbox_e->pack_start($etiqueta,0,0,0);
$etiqueta->show();

$entrada4 = new Gtk::Entry( 255 );
$entrada4->signal_connect( "changed", \&entrada_modificada4,
$entrada4 );
$entrada4->set_text( "/usr/mpi/lib" );
$vbbox->pack_start( $entrada4, 0, 0, 0 );
$entrada4->show();

$hbox_e= new Gtk::HBox(0, 2);
$hbox_e->border_width(3);
$vbbox->add($hbox_e);
$hbox_e->show();

$etiqueta = new Gtk::Label( "Parámetros Específicos (Extras):" );
$etiqueta->set_justify( 'left' );
$hbox_e->pack_start($etiqueta,0,0,0);
$etiqueta->show();
$entrada5 = new Gtk::Entry( 255 );

```

```

    $entrada5->signal_connect( "changed", \&entrada_modificada5,
$entrada5 );
    $entrada5->set_text( " " );
    $vbox->pack_start( $entrada5, 0, 0, 0 );
    $entrada5->show();

    # Separador
    $separador = new Gtk::HSeparator();
    $vbox->pack_start( $separador ,0,0,0);
    $separador->show();
    # creamos el empaquetado horizontal
    $hbox = new Gtk::HBox( 0, 3 );
    $vbox->add( $hbox );
    $hbox->border_width( 3 );
    $hbox->show();

    # Creamos el boton de generar.
    $boton = new Gtk::Button( " Generar " );
    $boton->signal_connect( "clicked", \&genera_dist);
    $hbox->pack_start( $boton, 0, 0, 0 );
    $boton->show();

    $boton_cancel = new Gtk::Button( " Cancelar " );
    $boton_cancel->signal_connect( "clicked", sub { $ventana-
>destroy(); } );
    $hbox->pack_start( $boton_cancel, 0, 0, 0 );
    $boton_cancel->show();

    $ventana->show();
    $ventana->grab_add();
}

```

Función que genera la lista de contadores así como todas su posibles combinaciones. La lista se extiende hasta 176 combinaciones, pero aquí solo se muestra una parte por el espacio requerido por la lista.

```

sub genera_lista_cont
{
    my $clist=$_[0];

    # Something silly to add to the list. 4 rows of 2 columns each.
    my @lista = ( [ L1DCACHE_MISS      => "L1ICACHE_READ" ],
                  [ L1DCACHE_MISS      => "L1ICACHE_MISS" ],
                  [ L1DCACHE_MISS      => "L2CACHE_READWRITE" ],
                  ...

```

```

...
...
[ JUMP_UNSUCCESS => "STALL" ],
[ JUMP          => "STALL" ],
[ MFLOPS        => "ELAPSED_CYCLES" ],
[ IPC           => "ELAPSED_CYCLES" ],
[ L1DCACHE_MISSRATE => "ELAPSED_CYCLES" ] );

for my $i ( 0..$#lista )
{
    $clist->append( @{$lista[ $i ]} );
}

return;
}

```

Está función permite obtener la información que se seleccione del cuadro donde se presenta la lista de contadores.

```

sub en_seleccion_cont
{
    my ( $clist, $renglon, $columna, $evento, @datos ) = @_;

    #my $texto;

    # Obtenemos el texto almacenado en el renglón y columna
    # en el que se hizo click. Lo recibimos con un apuntador en el
    # argumento de texto.
    #$texto = $clist->get_text( $row, $column );

    # Solo imprimimos alguna información sobre el texto
    open(TMP, "> .tmp_contadores_lib");
    print TMP "$renglon";
    close( TMP );
    #print( "\t\t$texto\n\n" );

    return;
}

```

Función que presenta al usuario la ventana donde puede seleccionar el o los contadores con los que puede seleccionar para generar el código perfilado.

```

sub contadores

```

```

{
    my $separador;
    my $ventana;
    my $vbox;
    my $hbox;
    my $ventana_scroll;
    my $clist;
    my $boton;
    my $boton_c;

    my $titles_visible = 1;
    my @titulos = ( "Contador 0", "Contador 1" );

    # Creamos la ventana
    $ventana = new Gtk::Window( "oplevel" );
    $ventana->set_usize( 403, 400 );
    $ventana->set_title( "Contadores" );
    $ventana->signal_connect( "destroy", sub { $ventana->destroy(); }
);

    $vbox = new Gtk::VBox( 0, 2 );
    $ventana->add( $vbox );
    $vbox->border_width( 3 );
    $vbox->show();

    # Creamos la ventana con scrolls para poder poner el widget de
    # clist en ella.
    $ventana_scroll = new Gtk::ScrolledWindow( undef, undef );
    #$vbox->pack_start( $ventana_scroll, 1, 1, 0 );
    $ventana_scroll->set_policy( 'automatic', 'always' );
    $ventana_scroll->set_usize( 220, 360 );
    $vbox->pack_start( $ventana_scroll, 0, 0, 0 );
    $ventana_scroll->show();

    # Creamos el CList
    $clist = new_with_titles Gtk::CList( @titulos );

    # Cuando hacemos una selección, necesitamos saber que fue lo que
    # seleccionamos.
    $clist->signal_connect( "select_row", \&en_seleccion_cont );

    # No es necesario ponerle sombra, pero se ve bonito :)
    $clist->set_shadow_type( 'out' );

    # Lo que si es importante, es poner el. Notamos que las columnas
    # son numeradas
    # de 0 en adelante.
    $clist->set_column_width( 0, 200 );
    $clist->set_column_width( 1, 200 );

    # Agregamos la lista a la ventana con scrolls y la mostramos.

```

```

$ventana_scroll->add( $clist );
genera_lista_cont($clist);
$clist->show();

# Creamos el separador para el boton y la lista.
$separador = new Gtk::HSeparator();
$vbox->add( $separador );
$separador->show();

# Cuadro de empaquetado horizontal
$hbox = new Gtk::HBox( 0, 2 );
$vbox->add( $hbox );
$hbox->border_width( 3 );
$hbox->show();

# Boton para cerrar la ventana.
$boton = new Gtk::Button( " Siguiente -> " );
$hbox->pack_start( $boton ,0, 0,0);
$boton->show();
$boton->signal_connect( 'clicked', \&genera_binario );

# Boton para la ayuda.
$boton_c = new Gtk::Button( " Cancelar " );
$hbox->pack_start( $boton_c ,0,0,0);
$boton_c->show();
$boton_c->signal_connect( 'clicked', sub { $ventana->destroy();
} );
$ventana->show();
$ventana->grab_add();
}

```

Se obtiene la información correspondiente a la selección de la carga a agregar a la lista.

```

sub seleccion_ok_dist
{
    my ( $widget, $seleccion_archivo ) = @_;
    my $archivo = $seleccion_archivo->get_filename();

    my @arreglo=split("/", $archivo);
    my $cadena=pop(@arreglo);
    if ( $cadena ne "<carga de trabajo>" )
    {
        system ("cp ".$archivo." cargas_dist/");
    }
    else

```

```

    {
        warn "Advertencia: No se a seleccionado un archivo.\n";
    }
}

```

Cuadro de selección de cargas que formaran parte de la lista de la aplicación.

```

sub selecciona_archivo_dist
{
    my $file_dialog;

    # Creamos un widget de selección de archivos
    $file_dialog = new Gtk::FileSelection( "Selección de archivos" );
    $file_dialog->signal_connect( "destroy", sub { $file_dialog->
destroy(); } );

    # Conectamos el boton de ok con la función file_ok_sel
    $file_dialog->ok_button->signal_connect(
"clicked", \&seleccion_ok_dist, $file_dialog );

    # Conenctamos el boton de calncel con la guncion de destrucción
del widget.
    $file_dialog->cancel_button->signal_connect( "clicked", sub
{$file_dialog->destroy();});

    # Vamos a ver el archivo, Como si fuéramos a salvar algun
archivo.., y
    # caulquer nombre que queramos darle por default lop onemos e
las comillas.
    $file_dialog->set_filename( "<carga de trabajo>" );

    $file_dialog->show();
    $file_dialog->grab_add();
}

```

Función principal de la interface de las cargas distribuidas, aquí se le presenta al usuario la ventana con la lista de las posibles cargas a seleccionar.

```

sub carga_distribuida
{
    my $separador;
    my $ventana;
    my $vbox=0;
    my $hbox;
    my $ventana_scroll;
    my $clist;
}

```



```

my $boton;
my $boton_cerrar;
my $boton_a;

my $titles_visible = 1;
my @titulos = ( "Trabajo Distribuido");

# Creamos la ventana
$ventana = new Gtk::Window( "toplevel" );
$ventana->set_usize( 220, 250 );
$ventana->set_title( "Selección de Carga" );
$ventana->signal_connect( "destroy", sub { $ventana->destroy();
} );

$ vbox = new Gtk::VBox( 0, 2 );
$ventana->add( $ vbox );
$ vbox->border_width( 3 );
$ vbox->show();

# Creamos la ventana con scrolls para poder poner el widget de
clist en ella.
$ventana_scroll = new Gtk::ScrolledWindow( undef, undef );
# $ vbox->pack_start( $ventana_scroll, 1, 1, 0 );
$ventana_scroll->set_policy( 'automatic', 'always' );
$ventana_scroll->set_usize( 200, 210 );
$ vbox->pack_start( $ventana_scroll, 0, 0, 0 );
$ventana_scroll->show();

# Creamos el CList
$clist = new_with_titles Gtk::CList( @titulos );
# Cuando hacemos una selección, necesitamos saber que fue lo que
seleccionamos.
$clist->signal_connect( "select_row", \&en_seleccion_dist );

# No es necesario ponerle sombra, pero se ve bonito :)
$clist->set_shadow_type( 'out' );

# Lo que si es importante, es poner el. Notamos que las columnas
son numeradas
# de 0 en adelante.
$clist->set_column_width( 0, 200 );

# Agregamos la lista a la ventana con scrolls y la mostramos.
$ventana_scroll->add( $clist );
genera_lista_dist($clist);
$clist->show();

# Creamos el separador para el boton y la lista.
$separador = new Gtk::HSeparator();
$ vbox->add( $separador );

```

```

$separador->show();

# Contenedor horizontal
$hbox = new Gtk::HBox( 0, 3 );
$vbox->add( $hbox );
$hbox->border_width( 3 );
$hbox->show();

# Boton para continuar la agregar..
$boton_a = new Gtk::Button( " Agregar " );
$hbox->pack_start( $boton_a,0,0,0 );
$boton_a->show();
$boton_a->signal_connect( 'clicked', \&selecciona_archivo_dist );

# Boton para continuar la ventana.
$boton = new Gtk::Button( " Siguiente -> " );
$hbox->pack_start( $boton,0,0,0 );
$boton->show();
$boton->signal_connect( 'clicked', \&contadores );

# Boton para cerrar la ventana.
$boton_cerrar = new Gtk::Button( " Cerrar " );
$hbox->pack_start( $boton_cerrar,0,0,0 );
$boton_cerrar->show();
$boton_cerrar->signal_connect( 'clicked', sub { $ventana->destroy(); } );

$ventana->show();
$ventana->grab_add();
}

```

Como estamos trabajando con un modulo el cual solo contiene funciones es necesario siempre regresar un valor por eso pones por omisión el uno.

```

1;
# Fin del programa

```

Módulo de control de las cargas no distribuidas.

Función que obtiene las cargas no distribuidas del directorio designado y

genera una lista.

```
sub genera_lista
{
    my $clist=$_[0];
    my @lista;
    unless(opendir(CDIR,"cargas_no_dist"))
    {
        print "No se puede leer el actual directorio\n";
    }
    my $i=0;
    foreach $archivo (readdir(CDIR))
    {
        if (($archivo ne ".") && ($archivo ne ".."))
        {
            $lista[0]=$archivo;
            $clist->append(@lista);
        }
    }
    closedir(CDIR);

    return;
}
```

Aquí se obtiene la información de la selección en la lista de aplicaciones.

```
sub en_seleccion
{
    my ( $clist, $row, $column, $event, @data ) = @_;

    my $texto;

    # Obtenemos el texto almacenado en el renglón y columna
    # en el que se hizo click. Lo recibimos con un apuntador en el
    # argumento de texto.
    $texto = $clist->get_text( $row, $column );

    # Solo imprimimos alguna información sobre el texto
    open(TMP,"> .tmp_nodist");
    print TMP "$texto";
    close( TMP );

    return;
}
```

Se selecciona una nueva carga a partir de un archivo, se incluye y se genera

el código perfilado de esa carga al dar la opción de ok.

```
sub seleccion_ok
{
    my $i;
    my ( $widget, $seleccion_archivo ) = @_;
    my $archivo = $seleccion_archivo->get_filename();

    my @arreglo=split("/", $archivo);
    my $cadena=pop(@arreglo);
    if ( $cadena ne "<carga de trabajo>" )
    {
        #print( "$archivo\n" );
        my @arreglo=split("/", $archivo);
        my $nombre=pop(@arreglo);
        #print $nombre."\n";

        open( ARCH, "< ".$archivo);
        my $cadena=<ARCH>;
        chomp ( $cadena );
        my @arr=split(" ", $cadena);
        my $funcion=pop(@arr);
        close( ARCH );
        print $funcion."\n";

        open(ARCH_D, "> cargas_no_dist/".$nombre);
        open(ARCH_H, "< generador/encabezado");
        foreach $i (<ARCH_H>)
        {
            print ARCH_D "$i";
        }
        close( ARCH_H );
        open(ARCH_S, "< ".$archivo);
        foreach $i (<ARCH_S>)
        {
            print ARCH_D "$i";
        }
        close( ARCH_S );
        open(ARCH_C, "< generador/cuerpo");
        foreach $i (<ARCH_C>)
        {
            print ARCH_D "$i";
        }
        close( ARCH_C );
        print ARCH_D "$funcion".";\n";
        open(ARCH_F, "< generador/pie");
        foreach $i (<ARCH_F>)
        {
            print ARCH_D "$i";
        }
    }
}
```

```

    }
    close( ARCH_F );
    close( ARCH_D );
    $seleccion_archivo->destroy();
}
else
{
    warn "Advertencia: No se a seleccionado ningun archivo.\n";
}
return;
}

```

Selección de archivo para incluirlo en la lista de aplicaciones disponibles, a partir de un dialogo de selección.

```

sub selecciona_archivo
{
    my $file_dialog;

    # Creamos un widget de selección de archivos
    $file_dialog = new Gtk::FileSelection( "Selección de archivos" );
    $file_dialog->signal_connect( "destroy", sub { $file_dialog->
destroy(); } );

    # Conectamos el boton de ok con la función file_ok_sel
    $file_dialog->ok_button->signal_connect(
"clicked", \&seleccion_ok, $file_dialog );

    # Conenctamos el boton de calncel con la guncion de destrucción
del widget.
    $file_dialog->cancel_button->signal_connect( "clicked", sub
{ $file_dialog->destroy(); } );

    # Vamos a ver el archivo, Como si fuéramos a salvar algun
archivo.., y
    # caulquer nombre que queramos darle por default lop onemos e
las comillas.
    $file_dialog->set_filename( "<carga de trabajo>" );

    $file_dialog->show();
    $file_dialog->grab_add();
}

```

Función que genera un ejecutable tomando en cuenta los datos dentro de los

archivos temporales previamente generados en este módulo.

```
sub genera
{
    my ($widget, $entrada)=@_;
    my $compila;
    my $cadena=0;
    unless(open (ARCH_TMP,"< .tmp_nodist"))
    {
        warn "Advertencia: No se a seleccionado la carga\n";
    }
    else
    {
        $cadena=<ARCH_TMP>;
        close( ARCH_TMP );
    }

    open(AR,"< .tmp_nodist_lib");
    my $texto=<AR>;
    close( AR );
    open (AR2,"< .tmp_nodist_extra");
    my $texto2=<AR2>;
    close( AR2 );
    if ($cadena)
    {
        $compila="gcc cargas_no_dist/" . $cadena . " -o bin/";
        chop($cadena);
        chop($cadena);
        $compila=$compila.$cadena." -L".$texto." -lpcl"." ".$texto2;
        system ($compila);
    }
    else
    {
        warn "Advertencia: No se a definido la carga para el
experimento\n\n";
    }
    return;
}
```

Funciones que obtienen la información de las rutas de las bibliotecas necesarias para generar código no distribuido perfilado.

```
sub entrada_modificada
{
    my ( $widget, $entrada ) = @_;
    my $entrada_texto = $entrada->get_text();
```



```

    open (TMP,"> .tmp_nodist_lib");
    print TMP "$entrada_texto";
    close( TMP );
    return;
}

sub entrada_extra
{
    my ( $widget, $entrada ) = @_;
    my $entrada_texto = $entrada->get_text();
    open (TMP,"> .tmp_nodist_extra");
    print TMP "$entrada_texto";
    close( TMP );
    return;
}

```

Interfase de las rutas de bibliotecas para las aplicaciones no distribuidas y captura la información.

```

sub ruta_libreria_pcl
{
    my $ventana;
    my $vbox;
    my $hbox;
    my $hbox_e;
    my $entrada;
    my $entrada2;
    my $boton;
    my $boton_cancel;
    my $etiqueta;
    my $separador;

    # Create a window
    $ventana = new Gtk::Window( "oplevel" );
    $ventana->set_usize( 210, 140 );
    $ventana->set_title( "Ruta de Bibliotecas" );
    $ventana->border_width(1);
    $ventana->signal_connect( "destroy", sub { $ventana->destroy();
} );

    $vbox = new Gtk::VBox( 0, 2 );
    $vbox->border_width(3);
    $ventana->add( $vbox );
    $vbox->show();

    $hbox_e= new Gtk::HBox(0, 2);
    $hbox_e->border_width(3);

```

```

$ vbox->add($ hbox_e);
$ hbox_e->show();

$ etiqueta = new Gtk::Label( "Ruta del PCL:" );
$ etiqueta->set_justify( 'left' );
$ hbox_e->pack_start($ etiqueta,0,0,0);
$ etiqueta->show();

# Creamos la entrada de texto.
$ entrada = new Gtk::Entry( 255 );
$ entrada->signal_connect( "changed", \&entrada_modificada,
$ entrada );
$ entrada->set_text( "/usr/local/lib" );
$ vbox->pack_start( $ entrada, 0, 0, 0 );
$ entrada->show();

# Rutas extras..
$ hbox_e = new Gtk::HBox(0, 2);
$ hbox_e->border_width(3);
$ vbox->add($ hbox_e);
$ hbox_e->show();

$ etiqueta = new Gtk::Label( "Parámetros Específicos (Extras):" );
$ etiqueta->set_justify( 'left' );
$ hbox_e->pack_start($ etiqueta,0,0,0);
$ etiqueta->show();

# Creamos la entrada de texto.
$ entrada2 = new Gtk::Entry( 255 );
$ entrada2->signal_connect( "changed", \&entrada_extra, $ entrada2
);
$ entrada2->set_text( " " );
$ vbox->pack_start( $ entrada2, 0, 0, 0 );
$ entrada2->show();

# Separador
$ separador = new Gtk::HSeparator();
$ vbox->pack_start( $ separador ,0,0,0);
$ separador->show();

# creamos el empaquetado horizontal
$ hbox = new Gtk::HBox( 0, 3 );
$ vbox->add( $ hbox );
$ hbox->border_width( 3 );
$ hbox->show();

# Creamos el boton de generar.
$ boton = new Gtk::Button( " Generar " );
$ boton->signal_connect( "clicked", \&genera,$ entrada );
$ hbox->pack_start( $ boton, 0, 0, 0 );
$ boton->show();

```

```

$boton_cancel = new Gtk::Button( " Cancelar " );
$boton_cancel->signal_connect( "clicked", sub { $ventana->destroy(); } );
$hbox->pack_start( $boton_cancel, 0, 0, 0 );
$boton_cancel->show();

$ventana->show();
$ventana->grab_add();
}

```

Interfase principal que muestra la lista de aplicaciones no distribuidas disponibles para el usuario, estas aplicaciones ya han sido perfiladas.

```

sub carga_no_distribuida
{
    my $separador;
    my $ventana;
    my $vbox_c;
    my $hbox;
    my $ventana_scroll;
    my $clist;
    my $boton;
    my $boton_gen;
    my $boton_cerrar;

    my $titles_visible = 1;
    my @titulos = ( "Trabajo No Distribuido");

    # Creamos la ventana
    $ventana = new Gtk::Window( "toplevel" );
    $ventana->set_usize( 225, 253 );
    $ventana->set_title( "Selección de Carga" );
    $ventana->signal_connect( "destroy", sub { $ventana->destroy(); } );

    $vbox_c = new Gtk::VBox( 0, 2 );
    $ventana->add( $vbox_c );
    $vbox_c->border_width( 3 );
    $vbox_c->show();

    # Creamos la ventana con scrolls para poder poner el
    widget de clist en ella.
    $ventana_scroll = new Gtk::ScrolledWindow( undef, undef

```

```

);
$ vbox_c->pack_start( $ ventana_scroll, 0, 0, 0 );
$ ventana_scroll->set_policy( 'automatic', 'always' );
$ ventana_scroll->set_usize( 200, 210 );
# $ vbox->add($ ventana_scroll);
$ ventana_scroll->show();

# Creamos el CList
$ clist = new_with_titles Gtk::CList( @titulos );

# Cuando hacemos una selección, necesitamos saber que
fue lo que seleccionamos.
$ clist->signal_connect( "select_row", \&en_seleccion );

# No es necesario ponerle sombra, pero se ve bonito :)
$ clist->set_shadow_type( 'out' );

# Lo que si es importante, es poner el. Notamos que las
columnas son numeradas
# de 0 en adelante.
$ clist->set_column_width( 0, 200 );

# Agregamos la lista a la ventana con scrolls y la
mostramos.
$ ventana_scroll->add( $ clist );
genera_lista($clist);
$ clist->show();

# Creamos el separador para el boton y la lista.
$ separador = new Gtk::HSeparator();
$ vbox_c->pack_start( $ separador ,0,0,0);
$ separador->show();

# Contenedor horizontal para poner los botones.
$ hbox = new Gtk::HBox( 0, 2 );
$ vbox_c->add( $ hbox );
$ hbox->border_width( 3 );
$ hbox->show();

# Boton para cerrar la ventana.
$ boton = new Gtk::Button( " Generar " );
$ hbox->pack_start( $ boton ,0,0,0);
$ boton->show();
$ boton->signal_connect( 'clicked', \&selecciona_archivo
);

```

```

    # Boton para generar na nueva carga no distribuida pero
instrumentada.
    $boton_gen = new Gtk::Button( " Siguiente -> " );
    $hbox->pack_start( $boton_gen ,0,0,0);
    $boton_gen->show();
    $boton_gen->signal_connect( 'clicked',
    \&ruta_libreria_pcl );

    $boton_cerrar = new Gtk::Button( " Cerrar " );
    $hbox->pack_start( $boton_cerrar ,0,0,0);
    $boton_cerrar->show();
    $boton_cerrar->signal_connect( 'clicked', sub {
$ventana->destroy();} );

    $ventana->show();
    $ventana->grab_add();
}

```

Como es un modulo que solo contiene funciones es necesario regresar siempre un valor, así que siempre regresamos 1 o verdadero no importa lo que suceda.

```

1;
# Fin del programa

```

Módulo de control de la ayuda.

En este módulo se genera la ventana con el manual de la aplicación.

```

sub ayuda_manual
{
    my $linea;
    my $ventana;
    my $principal_vbox;
    my $vbox;
    my $boton;
    my $separador;
    my $tabla;
    my $vscrollbar;
    my $texto;
    my $cmap;
}

```

```

my $color;

# Create a window
$ventana = new Gtk::Window( 'toplevel' );
$ventana->set_usize( 550, 500 );
$ventana->set_policy(1, 1, 0 );
$ventana->signal_connect( 'destroy', sub { $ventana->destroy();}
);
$ventana->set_title( "Ayuda Contadores" );
$ventana->border_width( 0 );

$principal_vbox = new Gtk::VBox( 0, 0 );
$ventana->add( $principal_vbox );
$principal_vbox->show();

$vbox = new Gtk::VBox( 0, 10 );
$vbox->border_width( 10 );
$principal_vbox->pack_start( $vbox, 0, 0, 0 );
$vbox->show();

# Create a table
$tabla = new Gtk::Table( 2, 2, 0 );
$tabla->set_row_spacing( 0, 2 );
$tabla->set_col_spacing( 0, 2 );
$vbox->pack_start( $tabla, 0, 0, 0 );
$tabla->show();

# Create the Text widget
$texto = new Gtk::Text( undef, undef );
$texto->set_editable( 1 );
$texto->set_usize(550,435);
$tabla->attach( $texto, 0, 1, 0, 1,[ 'expand', 'shrink', 'fill'
],[ 'expand', 'shrink', 'fill' ],0, 0 );

$texto->show();

# Add a vertical scrollbar to the GtkText widget
$vscrollbar = new Gtk::VScrollbar( $texto->vadj );
$tabla->attach( $vscrollbar, 1, 2, 0, 1, 'fill',[ 'expand',
'shrink', 'fill' ], 0, 0 );
$vscrollbar->show();

# Get the system color map and allocate the color red
$cmap = Gtk::Gdk::Colormap->get_system();
$color->{ 'red' } = 0;
$color->{ 'green' } = 0;
$color->{ 'blue' } = 0xFFFF;
unless ( defined( $cmap->color_alloc( $color ) ) )
{
    warn( "Couldn't allocate color\n" );
}

```



```

    # Realizing a widget creates a window for it, its now ready for
us to
    # insert some text
    $texto->realize();

    # Freeze the text widget, ready for multiple updates
    $texto->freeze();

    # Insert some colored text
    $texto->insert( undef, $color, undef, "Plataforma Intel Pentium
Pro, Pentium II, Pentium III, Pentium IV.\n\n" );

    # Load the file text (this program) into the text window
    open( FILE, "ayuda/ayuda_contadores.y" );

    foreach $linea ( <FILE> )
    {
        $texto->insert( undef, $texto->style->black, undef, $linea );
    }

    close( FILE );

    # Thaw the text widget, allowing the updates to become visible
    $texto->thaw();

    $texto->set_editable( 0 );

    $separador = new Gtk::HSeparator();
    $principal_vbox->pack_start( $separador, 0, 0, 0 );
    $separador->show();

    $vbox = new Gtk::VBox( 0, 10 );
    $vbox->border_width( 10 );
    $principal_vbox->pack_start( $vbox, 0, 0, 0 );
    $vbox->show();

    # Creamos el botonde cerrar.
    $boton = new Gtk::Button( "Cerrar" );
    $boton->signal_connect( 'clicked', sub { $ventana->destroy(); }
);
    $vbox->pack_start( $boton, 0, 0, 0 );
    #$boton->can_default( 1 );
    #$boton->grab_default();
    $boton->show();

    $ventana->show();
    $ventana->grab_add();
}

```

Como es un módulo que solo contiene una función es necesario regresar en

todo momento un valor en este caso verdadero o uno.

```
1;  
# Fin del Código.
```

Bibliografía

Raj Jain, 1991. "The art Of computer systems performance analysis. Techniques for experimental design, measurement, simulation and modeling".

Tom shaley, 199. "Pentium Pro and Pentium II system architecture", second edition. Mind Share INC.

William Gropp, Ewing Lusk y Anthony Skjellum, 1999. "Using MPI: Portable parallel programming with the message-passing interface", second edition. The MIT Press.

William Gropp, Ewing Lusk y Rajeev Thakur, 1999. "Using MPI 2: Advance features of message-passing interface", the MIT Press.

Neil Matthew, Richard Stones, 2000. "Professional Linux Programming", Wrox Press.

Eric Harlow, 1999. "Desarrollo de aplicaciones Linux con GTK y GDK", Pretince Hall.

Richard Peterson, 2001. "Linux programming a beginners guide", Osborne/McGraw Hill.

Hasan Gomaa, 2000. "Designing concurrent, Distributed and real time applications with UML", Addison-Wesley.

Marc Snir, Steve Otto, Steve Huss-Lederman, David Walker, Jack Dongarra, 1998. "MPI-The complete reference, Volumen 1. The MPI core", second edition. The MIT Press.

William Gropp, Steve Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, Marc Snir, 1998. "MPI_the complete reference, Volumen 2. The MPI-2 Extensions", The MIT Press.

Adan Hiraes Carbajal, 2000. "Evaluación de rendimiento de sistemas distribuidos basándose en un modelo geométrico de ejecución". CICESE.

Recursos en el WWW.

Tools for Profiling Library Management .

<http://www-fp.mcs.anl.gov/~lusk/papers/mpich-guide/node26.html>

<http://hpc.kangnung.ac.kr/docs/mpich-1.2.2.3/mpe/profiling/>

MPICH-A Portable Implementation of MPI.

<http://www-unix.mcs.anl.gov/mpi/mpich/>

MPICH Documents.

<http://www-unix.mcs.anl.gov/mpi/mpich/docs.html>

User's Guide for MPE: Extensions for MPI Programs.

<http://www-unix.mcs.anl.gov/mpi/mpich/docs/mpeguide/paper.htm>

User's Guide for mpich, a Portable Implementation of MPI.

<http://carpanta.dc.fi.udc.es/docs/mpi/mpich-userguide/paper.html>

MPE.

<http://www.clc.unibe.ch/cgi-bin/man2html?MPE+4>

PCL - The Performance Counter Library.

<http://www.fz-juelich.de/zam/PCL/>

TAU Evaluation.

<http://acts.nersc.gov/tau/evaluation.html>

TAU Portable Profiling.

<http://www.acl.lanl.gov/tau/download/subframedown.html>

Gt-Perl homepage.

<http://www.gtkperl.org/>

Gtk-Perl Tutorial.

<http://personal.riverusers.com/~swilhelm/gtkperl-tutorial/>

Clusters@top500.

<http://clusters.top500.org/>

Cluster Benchmarks Web Page.

<http://www.mfn.unipmn.it/~mino/cluster/benchmarks/>

IEEE Computer Society Task Force on Cluster Computing.

<http://www.ieeetfcc.org/>

Workload Characterization.

<http://www.caida.org/analysis/workload/>

Benchmark Resources.

<http://www.benchmarkresources.com/?Sel=Home>

Manuales y otras referencias.

Man pages "ar", 2002 Free Software Foundation, Inc.

Man pages "ranlib", 2002 Free Software Foundation, Inc.

Man pages "gcc", 2002 Free Software Foundation, Inc.