Simuladores completos de sistemas computacionales

Tomás Balderas Contreras
balderas@ccc.inaoep.mx

Instituto Nacional de Astrofísica, Óptica y Electrónica
Coordinación de Ciencias de la Computación
Ingeniería de Software

27 de noviembre, 2002

Contenido

Resumen

El proceso de desarrollo de sistemas complejos basados en computadoras, como los que hoy existen, requiere de herramientas que permitan caracterizar de forma precisa su comportamiento y rendimiento. La tecnología de *simulación completa de sistemas computacionales* (*complete instruction set simulation*) ha evolucionado a través de los años y ha producido programas muy sofisticados que permiten modelar diversas configuraciones de sistemas de cómputo, desde dispositivos empotrados de baja potencia hasta complejos multiprocesadores, supercomputadoras y redes de sistemas de cómputo. El uso de este tipo de herramientas se ha diversificado y en la actualidad son indispensables para los fabricantes de hardware para analizar sistemas complejos sin necesidad de construir prototipos, y para los ingenieros de software que requieren transportar programas a nuevas plataformas de hardware en proceso de desarrollo.

Palabras Clave: Simulación completa de sistemas de cómputo, sistemas operativos, intérpretes, traducción binaria, diseño y evaluación de computadoras.

1 Introducción

Un simulador completo de sistemas computacionales es un programa que reproduce el comportamiento de todos y cada uno de los componentes existentes en una computadora [2, 4, 9]. Si el simulador puede reproducir el comportamiento del CPU, la jerarquía de memoria y los dispositivos presentes en un máquina real entonces también podrá ejecutar los mismos programas, incluyendo al sistema operativo y el software de aplicación, sin necesidad de realizarles modificación alguna. El programa puede simular una gran variedad de arquitecturas de CPU (e.g. x86, IA-64, SPARC, MIPS, PA-RISC, Alpha, PowerPC, etc.), jerarquías de memoria (RAM, caché y gestión de memoria virtual) de diferentes tamaños y políticas de operación y una gran cantidad de dispositivos de propósitos diversos (e.g. controladores de video, interfaces de red, temporizadores, controladores de DMA, controladores de discos, etc.)

Las ventajas del uso de esta tecnología son varias y se mencionan a continuación:

1. Un simulador completo es necesario para el equipo de desarrollo de un nuevo sistema de cómputo, puesto que la información referente a su comportamiento recolectada por el simulador describe el funcionamiento de cada componente. Esta caracterización permite detectar errores de diseño y pro-

porcionar nuevas ideas para mejorarlos. También es mucho más económico verificar el comportamiento de un modelo de software que el de un prototipo de hardware.

2. Un simulador completo es conveniente para el desarrollo de software en diferentes plataformas. El sistema permite la interacción directa del usuario con múltiples tipos de computadoras sin necesidad de contar con el equipo real o de iniciar una estación de trabajo para conmutar a otro sistema operativo. Todas las máquinas simuladas están al alcance en el mismo ambiente de ejecución, lo que reduce costos y tiempo.

3. Un simulador completo es de gran utilidad para los ingenieros de software que requieren transportar tanto software de sistema como de aplicación a plataformas de hardware no existentes aún en el mercado y que se encuentran en proceso de diseño e implantación por el fabricante.

4. Un simulador completo es una herramienta excelente para proyectos de investigación en ambientes académicos. Sus ventajas son útiles para que los usuarios tengan una mejor comprensión del funcionamiento de una computadora y de la interacción entre el sistema operativo y el hardware.

Un aspecto de diseño muy importante que debe ser delimitado de forma precisa antes de iniciar el proceso de desarrollo es el nivel de detalle al que se desea simular los componentes del sistema real. Por un lado se requiere un programa de muy alto rendimiento y por otra parte necesitamos un modelo lo suficientemente preciso para ejecutar software sin modificaciones. La figura 1 ilustra

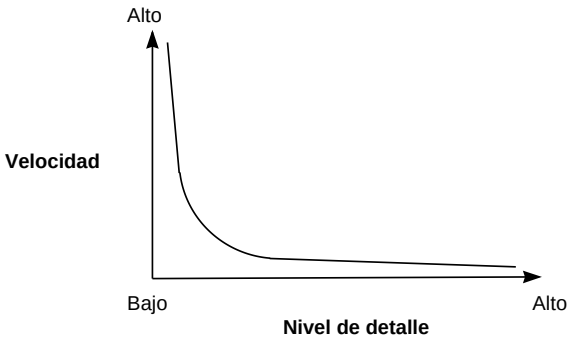

Figura 1: Relación entre el nivel de detalle del simulador y su rendimiento.

la relación que existe entre el nivel de detalle en el que se modelan los componentes y el rendimiento alcanzado por el simulador. El mejor compromiso entre estos dos factores consiste en simular los componentes a nivel de transferencia de registros. En este nivel de abstracción, el procesador debe ser simulado a nivel del conjunto de instrucciones, es decir, el modelo del CPU puede verse como un intérprete de instrucciones binarias de una arquitectura arbitraria X que se ejecuta en una arquitectura específica Y. En este nivel de abstracción se descartan aspectos de la lógica del CPU visto como un circuito integrado. Sin embargo, como deben ejecutarse varias instrucciones de la arquitectura Y por cada instrucción de la arquitectura X, inevitablemente un simulador de esta clase es varias veces más lento que el hardware real.

El objetivo de este reporte es describir los sistemas de simulación más sofisticados y completos que han sido desarrollados hasta la fecha, establecer los requerimientos que deben satisfacer estos programas y, finalmente, mencionar las historias de éxito de los grupos que han aplicado esta tecnología. El resto de este documento está organizado de la siguiente manera: la seccion 2 describe los requerimientos generales para los simuladores completos, la sección 3 proporciona información de un sistema europeo llamado Simics, la sección 4 describe el sistema SimOS desarrollado en la Universidad Stanford, en la sección 5 se comentan las aplicaciones exitosas que han tenido estos dos programas y, finalemente, se presentan las conclusiones de la investigación.

2 Requerimientos

Los requerimientos establecidos como resultado del proceso de investigación se clasifican en varias catagorías de acuerdo a: las funcionalidades proporcionadas por el sistema, la forma en que los usuarios pueden configurar el software, la portabilidad en el sentido de compilar el programa en distintos ambientes y simular distintas plataformas, la forma en que el usuario se comunica con el sistema y la capacidad del simulador para proporcionar al usuario la información que requiere.

2.1 Alto rendimiento y precisión

Todo proceso de diseño de sistemas basados en computadoras se beneficiaría de obtener información sobre el comportamiento tan rápido como sea posible. Mientras más rápido proporcione el simulador los datos sobre el rendimiento, más grande será el espacio de diseño que el desarrollador podrá evaluar en un espacio de tiempo específico. Por lo tanto el programa debe simular cada componente de la forma más rápida y eficiente posible, para lograrlo los componentes deben ser modelados a un nivel muy alto de abstracción.

El simulador debe ser lo suficientemente preciso para ejecutar la misma carga de trabajo que una computadora real. La carga incluye iniciar el sistema operativo, ejecutar múltiples procesos, entre programas de aplicación y de sistema, atención de llamadas al sistema, comunicación con los dispositivos de entrada/salida, etc. Más aún, el sistema operativo y el software de aplicación del

simulador son exactamente los mismos que los del hardware real, es decir, no debe ser necesario modificar los programas para que puedan ser ejecutados por el simulador.

El nivel de abstracción que ofrece la mejor combinación entre alto rendimiento y un buen grado de detalle es el nivel de conjunto de instrucciones, en el que el programa simula la ejecución de instrucciones una a la vez. Este nivel de abstracción no toma en cuenta aspectos de la lógica del circuito integrado, paralelismo a nivel de instrucciones y temporización.

2.2 Instrumentación

El principal beneficio de ejecutar cargas de trabajo en un simulador es que el estado completo de la computadora modelada está disponible para ser examinado. Este estado incluye los registros del CPU, los valores almacenados en todos los componentes de la jerarquía de memoria y los registros de los controladores de los dispositivos. Esta visibilidad permite al sistema detectar y registrar eventos que, aunque ocurren, no son registrados por el hardware de la computadora real.

La información estadística que puede ser obtenida en base al estado del hardware y a los eventos que en él ocurren debe ser recopilada, organizada y reportada al usuario en la forma que él estime más conveniente. Dicha información incluye:

- número de excepciones dentro de un programa
- número de accesos a memoria de datos o instrucciones por página de memoria virtual
- número de fallos/éxitos en el acceso a memorias caché
- número de fallos/éxitos en el acceso a páginas de memoria virtual
- efectividad de protocolos de coherencia en memoria caché
- número de instrucciones ejecutadas
- etc.

De especial relevancia es el hecho de que toda acción de recopilar información debe ser llevada a cabo sin interferir en la ejecución (*non-intrusive*). Por ejemplo, el simulador debe recolectar información sobre la frecuencia de éxitos y fallos en la memoria cahé sin influir en la forma en que esta lleva a cabo su tarea.

2.3 Determinismo

Un ambiente simulado es especialmente conveniente para reproducir el comportamiento del sistema real una y otra vez, esto es consecuencia del nivel de abstracción elegido para simular el sistema de cómputo. El programa, por lo tanto, debe comportarse de la misma manera cada vez que se encuentre en un determinado estado.

El simulador debe permitir al usuario reproducir fallas en el software o en el hardware y, además, debe proporcionarle toda la información al respecto, lo cual es difícil de llevar a cabo en un ambiente real sin interferir en su ejecución. Este requerimiento trae como consecuencia una reducción considerable en el espacio de búsqueda de errores que hayan sido detectados. Además, la ejecución determinista es especialmente útil en programas paralelos donde es común cometer errores al solucionar condiciones de competencia, las cuales son difíciles de duplicar.

Además de garantizar que el simulador se comporta de la misma forma cada vez que sus componentes alcanzan cierto estado, es deseable almacenar en un archivo (*checkpoint*) la información que conforma dicho estado, lo cual permite ahorrar tiempo en tareas repetitivas. Por ejemplo, para analizar el comportamiento de un sistema de gestión de bases de datos en diferentes sesiones es deseable comenzar cada sesión en el punto en que la base de datos está lista para operar, y saltar la secuencia de iniciación del sistema operativo y del gestor de bases de datos.

2.4 Extensibilidad

Un simulador debe acoplarse a las necesidades específicas de cada grupo de usuarios. Por esta razón un sistema de simulación debe aceptar componentes adicionales a los distribuidos con él inicialmente. Estos nuevos componentes pueden ser escritos ya sea por los usuarios, por el grupo que desarrolló el sistema o incluso por terceros, y se clasifican en dos grupos:

Dispositivos. Son modelos de dispositivos de entrada y salida, jerarquías de memoria, microcontroladores, etc. El usuario puede agregar un nuevo diseño para un dispositivo, examinar su comportamiento y mejorarlo hasta validarlo; o en su defecto puede agregar un modelo de un nuevo dispositivo en el mercado necesario para modelar de forma adecuada un sistema completo.

Extensiones. Son componentes que no modelan dispositivos sino que extienden la funcionalidad del sistema realizando alguna tarea específica. Por ejemplo comunicando información hacia y desde algún programa o interfaz externa al simulador, implantando módulos de análisis de la información recopilada, etc.

La herramienta principal que debe proporcionar el simulador a sus usuarios para escribir nuevos componentes es la API (*Application Programming Interface*), la cual debe ser robusta, clara y consistente. La API

contiene funciones, procedimientos, tipos de datos e interfaces predefinidas mediante los cuales el nuevo componente tiene acceso al estado del simulador y puede realizar todas sus operaciones.

Además de agregar componentes al sistema, el usuario debe ser capaz de especificar la forma en que el simulador responde a la ocurrencia de eventos; esto es posible si el programa proporciona mecanismos para asociar funciones con sucesos que ocurran en el ambiente simulado.

Otra forma de extender la funcionalidad del simulador es agregando comandos, cada comando puede estar asociado a algún dispositivo, a alguna extensión, o bien ser independiente. Los comandos también emplean la API y son añadidos mediante los mecanismos que el sistema debe definir.

Finalmente, es conveniente para el usuario agrupar los comandos y acciones necesarios para llevar a cabo una tarea compleja dentro de scripts, los cuales también pueden ser empleados para definir la configuración del sistema.

2.5 Simulación de diversas configuraciones

Entre las funcionalidades que el simulador debe proporcionar al usuario está la de permitirle configurar el modelo que desea simular. El simulador puede recibir un archivo de inicialización que especifica el número de procesadores a incluir en el modelo y la arquitectura (ISA) que los define, el tipo de cada dispositivo, la cantidad de memoria disponible, etc. Esta habilidad de configuración debe permitir al usuario modelar sistemas de las siguientes clases:

Sistemas empotrados (*embedded*): Son computadoras de propósito específico que contienen componentes que consumen poca potencia. Por ejemplo, PDAs, teléfonos celulares, cámaras inteligentes, etc.

Consolas de juegos de video: Son un tipo especial de sistemas empotrados que requieren de procesamiento intensivo y muchas capacidades para visualizar gráficos.

Estaciones de trabajo: Computadoras de propósito general con dispositivos de entrada y salida, capacidad de conexión en red, procesadores de arquitecturas CISC, RISC o EPIC.

Multiprocesadores: Son computadoras complejas con cientos o miles de unidades de procesamiento y con distintas arquitecturas de memoria (compartida, distribuida, etc.) y formas de interconección.

Debido a la gran flexibilidad que el simulador debe brindar, el usuario es libre de modelar un sistema que conste únicamente de un procesador y una jerarquía simplificada de memoria, sin niveles de memoria caché y sin dispositivos. Este tipo de configuración es conveniente para las personas interesadas en familiarizarse con la programación en lenguaje ensamblador de cierto microprocesador.

Actualmente es importante la comunicación de diversos dispositivos mediante una red. Por lo tanto, los sistemas de simulación deben proporcionar mecanismos para integrar las computadoras simuladas a la red local y, como consecuencia, modelar sistemas distribuidos y clusters para caracterizar su comportamiento.

2.6 Simulación de diferentes plataformas

Un requerimiento fundamental de un sistema simulador completo es su capaciadad de modelar sistemas basados en diferentes arquitecturas de microprocesador. Pueden existir varias distribuciones del mismo sistema para cada arquitectura (simulador estático), o bien, un solo programa que conforme a las necesidades del usuario cargue en tiempo de ejecución un modelo de CPU (simulador dinámico). En cualquier caso la forma en que el usuario trabaja con el sistema debe prevalecer, de tal forma que pueda migrar entre todas las sesiones con la seguridad de que todos reconocen los mismos comandos y proporcionan el mismo tipo de información.

Cuando se simulan procesadores con diferentes arquitecturas y se cuenta con los modelos de dispositivos adecuados, el usuario debe ser capaz de ejecutar una gran variedad de sistemas operativos y aplicaciones. La siguiente es una lista de algunas plataformas de hardware/software que debe ser posible simular:

1. (a) IA-32/Windows XP (b) IA-32/Linux
2. (a) SPARC/Solaris (b) SPARC/Linux
3. (a) PowerPC/MacOS X (b) PowerPC/Linux
4. MIPS/IRIX
5. Alpha/Tru64
6. PA-RISC/HP-UX
7. (a) IA-64/Linux (b) IA-64/HP-UX
8. POWER/AIX

El proceso de desarrollo de un simulador para una arquitectura específica no es una tarea trivial. Sin embargo, es posible diseñar herramientas para automatizar gran parte de dicho proceso.

2.7 Portabilidad

Además de simular distintas combinaciones de hardware y software, el simulador debe ser compilado en múltiples plataformas para incrementar su alcance. De esta forma el sistema se encontraría al alcance de un número mayor

de usuarios potenciales y les proporcionaría grandes posibilidades de desarrollo y productividad.

La tarea de portar un simulador puede requerir una gran inversión de tiempo. Generalmente el transportar el programa entre sistemas operativos basados en UNIX (Linux, Solaris, IRIX, BSD, etc.) no presenta mucha complicación pues la ABI del sistema operativo es idéntica y existen herramientas de desarrollo que están disponibles para todos estos sistemas operativos (e.g. *gcc*, *make*, *gdb*, etc.) Por lo tanto la compilación puede ser casi inmediata. Sin embargo, el transportar el simulador entre ambientes completamente distintos (e.g. NeXTSTEP y Windows) requiere un proceso de adaptación del código fuente que incluso puede influir en el código del simulador para los ambientes a los que ya había sido adaptado.

La figura 2 ilustra una plataforma de hardware y software (PA-RISC/HP-UX) en la que se ha compilado y ejecutado un simulador. Es deseable que el sistema se encuentre disponible para todas las plataformas existentes en el mercado. Sin embargo, este es una situación que aún no ha ocurrido, ni siquiera con los simuladores más sofisticados

2.8 Seguridad

El equipo de desarrollo del sistema puede, bajo su consideración, establecer políticas de uso y distribución del programa. El objetivo de estas reglas es proteger su inversión en investigación y desarrollo y proporcionar medidas de seguridad sobre la información de los usuarios.

Si el simulador fue desarrollado por una compañía de capitales privados entonces habrá un interés en restringir la ejecución del programa a los usuarios, dominios y máquinas registradas mediante un esquema de licencias flotantes. Sin embargo, también es posible distribuir el sistema mediante un esquema de software libre en el que cualquier usuario tiene la posibilidad de mejorarlo.

El sistema puede hacer distinción entre diferentes tipos de usuarios y, en base a ello, asignarles diferentes permisos de uso del sistema. Pero también es factible y deseable que todas las funcionalidades del programa estén al alcance de cualquier persona interesada sin distinción alguna por su experiencia, grado académico y tipo de proyecto.

2.9 Validación

La utilidad de un simulador completo depende completamente de que haya sido diseñado adecuadamente, de que sea eficiente y, sobre todo, de que realmente simule el comportamiento del hardware real. Un requerimiento básico es el de garantizar que el simulador reproduce fielmente el comportamiento de, por ejemplo, un procesador comercial.

Una buena estrategia durante el proceso de control de calidad del programa es desarrollar herramientas que contrasten los resultados de una o varias instrucciones ejecutadas sobre el mismo estado tanto en el simulador como en el procesador real, y que reporten cualquier diferencia que pueda ser ocasionada por una falla en el modelado. Cada prueba de este tipo es trivial, sin embargo se requieren millones de pruebas y considerar todas las posibles instrucciones que puede ejecutar el procesador. El desarrollo de estas herramientas se dificulta un poco al considerar conjuntos complejos de instrucciones e instrucciones privilegiadas. Además es necesario validar el resto de los modelos de los dispositivos de forma similar. Una propuesta para realizar tales pruebas se comenta en [3] para el simulador SimpleScalar.

El rendimiento del sistema es importante y puede ser estimado empleando los mismos recursos disponibles para computadoras reales, entre estos recursos resaltan los *benchmarks*, tales como SPEC y Dhrystone.

2.10 Interfaz de usuario

Uno de los componentes más importantes de un sistema de simulación es la interfaz al usuario, pues es el medio por el cual el usuario solicita al programa la información que necesita. La interfaz debe ser completamente funcional, clara y consistente entre todas las versiones del programa y entre los comandos tanto para el CPU como para los dispositivos y la memoria.

Una opción es proporcionar una interfaz de línea de comandos en la que la información solicitada se presenta una a la vez, de tal forma que el usuario recibe la información que desea y sólo esa. Este tipo de interfaz es muy útil para los desarrolladores que están acostumbrados a emplear herramientas completas de depuración tales como *gdb*. También es posible contar con una interfaz gráfica en donde el usuario pueda consultar en cualquier momento los datos que necesita y en donde pueda visualizar todos los comandos que le ofrece el sistema.

Es deseable proporcionar los medios que le permitan al usuario extender la funcionalidad de la interfaz. En un ambiente de línea de comandos, por ejemplo, es necesario contar con un lenguaje de procesamiento de scripts (e.g. Python) cuya tarea es ejecutar las ordenes plasmadas en el script. Para cualquier tipo de interfaz de usuario es posible incluir interfaces gráficas para controlar ciertos dispositivos, por ejemplo sistemas de adquisición de imágenes, adaptadores gráficos, etc.

2.11 Documentación

Los usuarios del sistema deben recibir, por lo menos, la siguiente documentación:

Guía del usuario: Es un documento que incluye información sobre el manejo básico del sistema, la instalación, los comandos genéricos, la forma de añadir

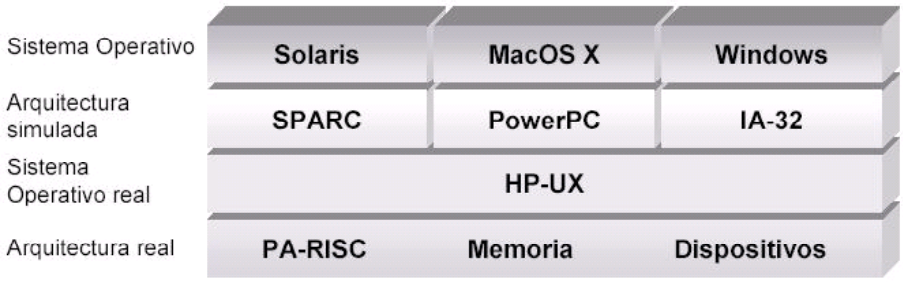

Figura 2: Portabilidad de un sistema de simulación.

extensiones, la descripción de todas las funcionalidades disponibles, la forma de solucionar problemas (*troubleshooting*) y los medios para adquirir soporte técnico por parte del fabricante. Esta guía no debe contener información dependiente de la plataforma que simula el programa, por lo tanto es un documento genérico e independiente de la versión del simulador.

Manual de referencia: Es un documento que contiene la descripción de cada comando, las opciones que puede procesar el sistema, las funciones, interfaces y tipos de datos proporcionados por la API para escribir extensiones y dispositivos y otro tipo de información más específica. El manual de referencia debe contener información específica de la plataforma que simula el programa y, por lo tanto, existirá un documento de este tipo para cada versión del simulador.

Adicionalmente, el programa debe contar con un asistente que proporcione información en línea a petición del usuario. Otro tipo de documentación que puede ser conveniente al usuario es la siguiente:

Tutoriales: Asistentes interactivos que proporcionan una visión general sobre las funcionalidades del programa y la forma de llevar a cabo algunas operaciones.

***Whitepapers*:** Son documentos que proporcionan información actualizada e introductoria acerca del sistema, de la tecnología subyacente y sus aplicaciones, o bien acerca del fabricante.

Publicaciones: Si el sistema es producto de un proceso de investigación y desarrollo en alguna institución de educación superior o centro privado de investigación, es muy posible que los desarrolladores hayan publicado artículos en memorias de congresos y revistas especializadas, reportes técnicos y tesis. Tales documentos contienen detalles sobre la implantación del sistema.

3 Simics

Simics es un simulador completo desarrollado inicialmente como un proyecto de investigación en el SICS (Swedish Institute of Computer Science). Desde 1998 ha sido extendido y portado a varias plataformas en Virtutech AB, una compañía con base en Estocolmo, Suecia.

Simics puede ser ejecutado, por lo menos, en plataformas SPARC/Solaris, x86/Windows, x86/Linux, Alpha/Tru64 y PowerPC/Linux. Puede simular computadoras basadas en cualquiera de las siguientes arquitecturas de procesadores: Alpha, ARM, IA-64, MIPS, PowerPC, SPARC y x86; también soporta la arquitectura x86-64, cuyos primeros microprocesadores aún están en fase de desarrollo. A cada arquitectura está asociado un conjunto amplio de simuladores de dispositivos que permiten construir un modelo virtual completo. Simics permite simular sistemas empotrados (embedded), sistemas con un solo procesador, sistemas multiprocesadores y, mediante el programa Simics Central, redes o clusters completos de máquinas simuladas [8].

Simics es un simulador de alto rendimiento que proporciona varias herramientas necesarias para su aplicación en tareas de diseño de microprocesadores, estudios de jerarquías de memoria, desarrollo de dispositivos, depuración, transporte de sistemas operativos y aplicaciones a arquitecturas no disponibles en el mercado (e.g. x86-64), etc.

Este simulador proporciona a sus usuarios una extensa API que les permite escribir extensiones al sistema y módulos que simulen nuevos dispositivos o componentes, con la finalidad de extender la funcionalidad del programa o verificar el comportamiento de nuevos diseños de hardware o software. La interfaz con el usuario es a través de línea de comandos, con la posibilidad de añadir nuevos comandos asociados a nuevos módulos en el sistema.

El desarrollo de Simics ha sido básicamente un proceso incremental que inició en el SICS con el desarrollo del simulador gsim [7], el cual es una extensión al simulador g88 que modela un multiprocesador basado en el procesador M88000 [2]. En 1994 comenzó un proyecto ambicioso de desarrollo y perfeccionamiento de esta tec-

nología de simulación, el cual produjo el sistema SimICS/sun4m que modela sistemas basados en la arquitectura SPARC. Algunas herramientas auxiliares en el proceso de implantación del simulador, que son utilizadas hasta la fecha, fueron desarrolladas en este periodo, tal es el caso de SimGen. SimGen es un programa que recibe, entre otras cosas, un archivo de especificación de una arquitectura de conjunto de instrucciones y genera automáticamente un intérprete, un ensamblador y un desensamblador para dicha arquitectura [6].

La siguiente etapa en el desarrollo de Simics comienza en 1998 con la fundación de Virtutech. A partir de entonces el equipo de desarrolladores crece, se automatizan varios procesos de prueba de rendimiento al simulador y de integración de su documentación, el sistema es extendido para simular otras arquitecturas, se porta a varias plataformas, se desarrollan lenguajes que faciliten la interacción entre el usuario y el sistema, se añaden varias funcionalidades nuevas, etc. La compañía ha logrado abrir un nuevo mercado para esta tecnología, que consta de firmas importantes (Intel, Sun Microsystems, AMD, etc.) y de universidades prestigiadas (CMU, MIT, etc.) Otro aspecto importante es la validación del software; el autor desarrolló en la compañía herramientas propias para analizar y comparar el comportamiento de los simuladores contra el de las computadoras reales con la finalidad de encontrar, de manera rápida y precisa, incongruencias en los sistemas que eventualmente deben ser corregidas. La figura 3 muestra varias sesiones del simulador que modelan distintas plataformas.

Simics simula instrucciones mediante un intérprete, generado automáticamente por la herramienta SimGen, que ejecuta instrucciones de forma secuencial y una a la vez. Recientemente la compañía incorporó modelos detallados de tiempo y ejecución fuera de orden al sistema.

4 SimOS

El proyecto SimOS comenzó en 1992 en la Universidad Stanford con el objetivo de construir una herramienta capaz de estudiar el comportamiento de la ejecución de sistemas computacionales modernos [4, 10]. La versión incial de SimOS modelaba computadoras con un solo procesador y con múltiples procesadores de arquitectura MIPS, que podían iniciar y ejecutar el sistema operativo IRIX de SGI. SimOS es compatible de forma directa con los sistemas de SGI y puede ejecutar casi cualquier programa diseñado para estas plataformas. Una versión reciente de SimOS simula computadoras basadas en la arquitectura Alpha con suficiente detalle, IBM por su parte desarrolló otra versión que modela computadoras basadas en la arquitectura PowerPC.

SimOS proporciona un enfoque modular para la simulación pues incluye interfaces bien definidas para el desarrollo y adición de múltiples CPUs, jerarquías de memoria y modelos de dispositivos de entrada y salida. Cada modelo proporciona la funcionalidad requerida para ejecutar el software, además modela detalles de la implantación del hardware que pueden variar. El objetivo de este enfoque es proporcionar al usuario la facilidad de configurar el nivel de detalle de simulación y velocidad que más le convenga. Para lograr este objetivo el simulador proporciona varios modelos de hardware compatibles que implatan diferentes técnicas de simulación para producir tres modelos principales.

Una diferencia fundamental entre los simuladores comentados anteriormente es la forma en que implantan ls ejecución de instrucciones. Simics utiliza un intérprete y una representación intermedia de las instrucciones. SimOS emplea el método de traducción binaria, que consiste en traducir cada instrucción de un programa ejecutado en el simulador en una secuencia de instrucciones del procesador que ejecuta el sistema.

5 Historias de éxito

En esta sección se describen muy brevemente algunos de los proyectos de investigación y desarrollo que han utilizado la tecnología de simulación completa y los resultados arrojados. El lector interesado en conocer los detalles puede consultar las referencias que se proporcionan.

5.1 Diseño del multiprocesador FLASH

SimOS fue ampliamente usado como auxiliar en el diseño del sistema FLASH en la universidad Stanford. La meta del grupo de desarrollo de FLASH fue construir un multiprocesador de memoria compartida capaz de escalar a miles de nodos de procesamiento basados en el procesador MIPS R10000.

El sistema de memoria propuesto para FLASH fue modelado y agregado a SimOS como una extensión (FLASHLite). Esto trajo como consecuencia la posibilidad de evaluar los diseños del multiprocesador con un gran número de cargas de trabajo. Los resultados incluyen la detección de irregularidades en el protocolo para coherencia de cachés en un ambiente de simulación operando en condiciones similares a las del hardware real en su vida útil.

5.2 Reconocimiento de habla

La versión de SimOS (SimOS-PPC) desarrollada en el centro de investigación de IBM en Austin fue empleada para caracterizar el proceso de reconocimiento de habla y para evaluar si los microprocesadores y jerarquías de memoria actuales son adecuados para este tipo de aplicaciones [1].

En este proyecto se caracterizó un sistema de reconocimiento basado en el programa de procesamiento de señales RASTA y el programa SPHINX que toma los

Figura 3: El simulador Simics (cortesía de Virtutech AB).

resultados del anterior para realizar una búsqueda en un grafo muy extenso. Los resultados reportados muestran que las jerarquías estudiadas no favorecen un buen rendimiento del software SPHINX y se proponen otros esquemas de memoria para solucionar los problemas.

5.3 Linux en x86-64

x86-64 es una arquitectura de conjunto de instrucciones concebida por AMD como una extensión a la popular arquitectura IA-32, la extensión incluye direcciones y datos de 64 bits. Las primeras implantaciones de esta arquitectura (Hammer) forman la nueva generación de procesadores de AMD.

En enero del 2001 Virtutech y AMD anunciaron conjuntamente la liberación de una versión de Simics que simula computadoras basadas en x86-64. Esta versión del simulador ha sido empleada por otras compañías para desarrollar software que estará disponible cuando el nuevo procesador sea liberado al mercado. Una de estas empresas es SuSE, que ha transportado Linux a x86-64 mediante un ambiente de simulación [5].

6 Conclusiones

Se describió de manera muy general la tecnología de simulación completa de computadoras, la cual permite modelar cualquier sistema de hardware con el nivel de detalle adecuado para lograr un alto rendimiento. Se establecieron los requerimientos que estos sistemas deben satisfacer, los cuales fueron identificados a través de una extensa revisión bibliográfica y en los comentarios de planes y proyectos expuestos por los principales innovadores de este tipo de programas. Se describieron las principales características de dos de los sistemás más sofisticados que han sido desarrollados hasta la fecha y, finalmente, se comentaron algunas aplicaciones.

El desarrollo de ambientes de simulación ha estado motivado por la creciente necesidad de modelar sistemas reales y obtener información sobre su comportamiento. Tanto los desarrolladores de hardware como los ingenieros de software se benefician de esta tecnología pues les permite obtener datos que son difíciles de obtener en un sistema real, caracterizar el comportamiento de programas de aplicación de alto nivel y desarrollar software para plataformas de hardware no disponibles.

Referencias

[1] Agaram, K., S. W. Keckler y D. Burger. 2001. A Characterization of Speech Recognition on Modern Computer Systems. En *Proc. of the 4th Annual International Workshop in Workload Characterization.*

[2] Bedichek, R. C. 1990. Some efficient architecture simulation techniques. En *Proc. of Winter 1990 USENIX Conference.*

[3] Glamn, B. y D. J. Lilja. 2000. Automatic Verification of Instruction Set Simulation Using Synchronized State Comparison. Reporte Técnico: ARCTiC-00-10. University of Minnesota, Department of Electrical and Computer Engineering, Minnesota, EUA.

[4] Herrod, S. A. 1998. Using Complete Machine Simulation to Understand Computer System Behavior. PhD Thesis. Reporte Técnico: CS-TR-98-1603. Stanford University, Department of Computer Science, California, EUA.

[5] Kleen, A. Porting Linux to x86-64. 2001. En *Proc. of the 2001 Linux Symposium.*

[6] Larsson, F., P. S. Magnusson y B. Werner. 1997. SimGen: Development of Efficient Instruction Set Simulators. Reporte Técnico: R97:03. Swedish Institute of Computer Science, Kista, Suecia.

[7] Magnusson, P. S. 1992. Efficient simulation of parallel hardware. Masters Thesis. Kungliga Tekniska Högskolan (Royal Institute of Technology), Estocolmo, Suecia.

[8] Magnusson, P. S., M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt y B. Werner. 2002. Simics: A Full System Simulation Platform. *IEEE Computer* 35(2):50–58.

[9] Magnusson, P. S., F. Dahlgren, H. Grahn, M. Karlsson, F. Larsson, F. Lundholm, A. Moestedt, J. Nilsson, P. Stenström y B. Werner. 1998. SimICS/sun4m: A Virtual Workstation. En *Proc. of the Usenix Annual Technical Conference.*

[10] Rosenblum, M., S. A. Herrod, E. Witchel y A. Gupta. 1995. Complete Computer System Simulation: The SimOS Approach. *IEEE Concurrency* 3(4):34–43.