Herramientas para la extensibilidad en entornos de ejecución paralelos

Albert Serra, Xavier Martorell, Ernest Artiaga,
Yolanda Becerra, Marisa Gil, Nacho Navarro

Abstract— **Mejorar el rendimiento de las aplicaciones paralelas ha supuesto considerables esfuerzos para adaptar los entornos de ejecución a sus necesidades. Una de las vertientes de esta adaptación es la especialización de los servicios que requieren estas aplicaciones. Para permitir que las propias aplicaciones aporten mejoras a los servicios básicos resulta conveniente disponer de mecanismos de soporte a la inserción dinámica de extensiones. En este artículo presentamos el estado de nuestra investigación en el campo de la extensibilidad del entorno de ejecución, así como las herramientas que hemos desarrollado para dar soporte a la experimentación. Actualmente, estas herramientas están siendo utilizadas en la investigación en computación de altas prestaciones ya que permiten obtener de forma sencilla información de alto nivel de la ejecución de las aplicaciones en condiciones reales, e intervenir en la ejecución mediante extensiones incrementales del entorno.**

Keywords— **sistemas operativos, extensibilidad, interposición, especialización**

I. Introducción

Mejorar el rendimiento de las aplicaciones paralelas ha supuesto considerables esfuerzos para adaptar los entornos de ejecución (librerías, servidores de sistema, kernel, ...) a sus necesidades. Una de las vertientes de la adaptación es la *especialización* de los servicios del entorno de ejecución de estas aplicaciones.

Una aproximación a la especialización consiste en ampliar el repertorio de implementaciones de cada servicio ofrecido por el sistema, y permitir que el usuario, la aplicación, el compilador, o incluso el sistema seleccionen la más adecuada en cada caso.

Alternativamente, puede permitirse que sean las propias aplicaciones las que aporten mejoras a los servicios básicos, para lo cual es conveniente disponer de mecanismos que permitan la inserción dinámica de extensiones.

Hasta ahora, los intentos de conseguir sistemas dinámicamente extensibles han dado lugar a mecanismos de carga dinámica de código, tanto dentro del kernel como en el espacio de usuario, así como a organizaciones del sistema operativo más adaptables (como los microkernels), y a mecanismos más flexibles de enlace entre clientes y servicios.

A pesar de estos avances, los sistemas operativos están aún lejos de proporcionar una extensibilidad adecuada a los propósitos citados. Los mecanismos que se utilizan para la selección de las implementaciones no son uniformes (variables de entorno, llamadas al sistema, ...), el enlace con los servicios, aunque dinámico, sigue bastante ligado al tiempo de generación del ejecutable, y el coste de la flexibilidad "estilo microkernel" es demasiado elevado para emplearlo en la computación de altas prestaciones. No en vano, la extensión del kernel por parte de las aplicaciones, los mecanismos de comunicación entre procesos, la reconfiguración del entorno de ejecución, o la propia estructura del kernel siguen siendo motivo de investigación.

En este artículo presentamos el estado de nuestra investigación en el campo de la extensibilidad del entorno de ejecución. Nuestro punto de vista es que la extensibilidad no solamente resulta útil para mejorar el rendimiento, sinó que las extensiones pueden aplicarse para fines tan diversos como monitorización, depuración, evolución del entorno (ampliación de servicios, "parcheo", ...), control de acceso, contabilización, e incluso para reestructurar el sistema.

En la siguiente sección describimos la aproximación que seguimos. A continuación, presentamos la herramienta que hemos desarrollado para dar soporte a nuestros experimentos. Seguidamente, mostramos algunos de los experimentos que hemos realizado. Les siguen algunas aplicaciones a la computación paralela, y, por último, a título de conclusión, hacemos un breve sumario del trabajo presentado.

II. Extensibilidad mediante interposición

Nuestra aproximación a la extensibilidad tiene como objetivo integrar en el entorno de ejecución los mecanismos necesarios para permitir *extensibilidad dinámica* (entendida como ampliación de la funcionalidad en tiempo de ejecución) sobre sistemas operativos actuales, sin afectar al funcionamiento de las aplicaciones ya existentes. Ésto

Este trabajo ha sido soportado por el Ministerio de Educación bajo el contrato TIC98-0511, y por el Comissionat per a Universitats i Recerca de la Generalitat de Catalunya

Los miembros del grupo pertenecen al Departament d'Arquitectura de Computadors de la Universitat Politècnica de Catalunya. Sus direcciones de correo son: {`alberts, xavim, ernest, yolandab, marisa, nacho`}`@ac.upc.es`

obliga a buscar mecanismos transparentes, que no pongan en peligro la compatibilidad con los sistemas actuales y que sean capaces de aprovechar las facilidades que proporcionan dichos sistemas.

En general, las extensiones van a ser una combinación de código y datos, con diversas *definiciones* de servicios accesibles desde el exterior, y conteniendo *referencias* a servicios implementados fuera de ellas.

Para enlazar las extensiones al entorno, nos inclinamos por utilizar *interposición*, es decir, inserción del código de las extensiones en las fronteras definidas por las interfaces de los servicios. Ésta es una técnica utilizada con relativa frecuencia para extender el sistema operativo (véanse [3] y [4], por ejemplo). Entre sus ventajas cabe citar que se trata de un mecanismo transparente y que, al no alterar la interfaz, permite ser aplicada recursivamente.

El ámbito de la extensión queda definido por el dominio del cual obtiene los recursos de ejecución. En el caso de extensiones introducidas por las aplicaciones, éstas serán privadas a la aplicación y su duración quedará ligada a la vida de la aplicación.

El hecho de respetar las interfaces no debe interpretarse como una restricción severa a la aplicación de las extensiones. En la figura 1 mostramos cómo la interposición se puede utilizar para invocar extensiones que no estan necesariamente ligadas a esta interfaz.

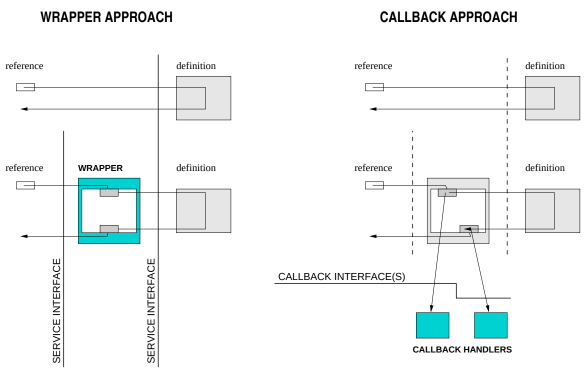

Fig. 1. Wrappers y callback handlers

En la figura se muestran dos alternativas para enlazar las extensiones (en color más oscuro) al entorno: a la izquierda, mediante un *service wrapper*, y a la derecha usando *service callback handlers*. Una extensión introducida mediante un *wrapper* se invoca en lugar de la definición convencional del servicio afectado, es decir, que el *wrapper* debe proporcionar la misma interfaz que el servicio extendido, y queda bajo su responsabilidad invocar a la forma original del servicio. En cambio, una extensión introducida mediante un *callback handler* se invoca "lateralmente" a la definición convencional del servicio, usando una interfaz independiente de la que está siendo extendida.

La implementación que describimos en la siguiente sección demuestra este modelo puede ser soportado por un sistema operativo comercial.

III. DITools para sistemas ELF/UNIX

Las DITools (Dynamic Interposition Tools) [9] son un conjunto de herramientas que hemos diseñado y desarrollado para permitir que los programas se puedan extender dinámicamente con funcionalidad no prevista mediante interposición. Las DITools se basan en la ABI (Application Binary Interface) ELF, soportada por muchos sistemas operativos modernos, como por ejemplo las últimas versiones de HP-UX, Linux, o IRIX.

Los ejecutables y las librerias ELF contienen estructuras de datos para permitir enlace dinámico y código independiente de posición. Un componente del sistema, llamado *enlazador dinámico*, utiliza estas estructuras durante la carga del programa y para resolver las referencias entre módulos.

El núcleo de las DITools –llamado DIRuntime– es una extensión del *enlazador dinámico* que exporta servicios para manipular las estructuras de datos mencionadas. Las extensiones de los servicios se proporcionan en módulos independientes –DIBackend– que solamente se cargan cuando la configuración activa lo requiere.

En esta implementación, hemos optado por utilizar la organización y el formato de las librerías de enlace dinámico para encapsular las extensiones. Ello nos permite reutilizar todas las herramientas de desarrollo ya disponibles.

Para introducir las extensiones en el espacio de direcciones de la aplicación utilizamos los servicios de carga dinámica proporcionados por el sistema operativo. El enlace de las extensiones se puede controlar con granularidad de función, modificando la política de resolución de referencias por defecto (la cual asocia una referencia con la definición que tiene el mismo nombre).

La figura 2 representa un programa paralelo que ha sido extendido con funcionalidad de monitorización usando DITools. Se distinguen los siguientes elementos: el *ejecutable principal* (arriba a la izquierda), el *runtime* de soporte al paralelismo (arriba a la derecha), el DIRuntime (abajo a la izquierda) y el DIBackend (abajo en el centro). El rectángulo superior contiene los componentes que se cargan normalmente cuando el programa es ejecutado, mientras que el rectángulo inferior encierra aquellos objetos que han sido cargados debido a que el programa está siendo extendido. Aparecen dos de las tablas mencionadas: la del programa principal, y la del DIBackend. La configuración activa

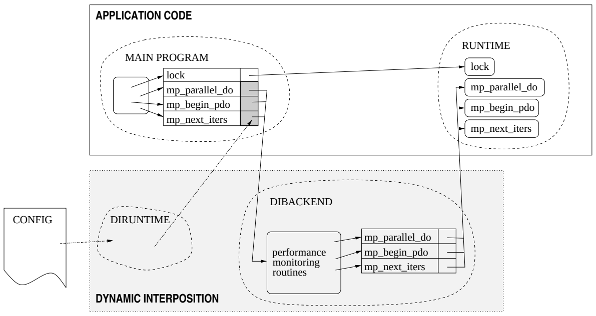

Fig. 2. Ejemplo de interposición mediante DITools

está descrita en un fichero adicional.

La misión del DIRuntime es redireccionar las referencias a servicios hacia las definiciones proporcionadas por el DIBackend, de acuerdo con la configuración activa.

El DIBackend puede referenciar las definiciones originales de los servicios, si es necesario.

IV. Experimentos y resultados

En esta sección mostramos dos experimentos en los que hemos utilizado la interposición para introducir dinámicamente extensiones en el entorno. En el primero de ellos, el objetivo es especializar algunos servicios frente a la presencia de un cierto tipo de reuso. En el segundo, el objetivo es introducir funcionalidad de monitorización en aplicaciones paralelas. En ambos casos, las extensiones son transparentes para las aplicaciones, y no es necesario regenerar ni los ejecutables ni las librerías implicadas.

A. Especialización de servicios del entorno

Para mostrar cómo es posible especializar el entorno de ejecución mediante extensiones, hemos empleado las ideas descritas con el fin de mejorar ciertos servicios usados por programas científicos, aplicando una técnica conocida como *memoization*.

Mediante la aplicación de esta técnica se pretende evitar la realización de cálculos redundantes, utilizando para ello una "cache" de resultados (véase [8] para más información). Esto es fácilmente implementable mediante una extensión estilo *wrapper* como la siguiente:

```
ret_type memoization_wrapper(arg_type arg)
{
  h = hash(arg);
  if (cache[h].argument != arg) {
    /* miss */
    cache[h].argument = arg;
    cache[h].result = service(arg);
  }
  return cache[h].result;
}
```

En la figura 3 se representa el efecto de esta especialización aplicada a diferentes servicios (en este caso funciones matemáticas) en varias aplicaciones. La plataforma en la cual han sido obtenidos estos resultados es un multiprocesador Intel Alder con el sistema operativo Linux. El compilador y el enlazador utilizados son de GNU (versión 2.7.2.3), así como las librerías de soporte. Hemos ejecutado aplicaciones científicas tomadas de SPLASH-2 [2] (FMM, WATER, FFT, OCEAN) y SPEC-95 (TURB3D, SWIM). La aplicación GMW proviene de un proyecto de optimización, y ROOM es un kernel de procesado de audio. Los tiempos que se muestran son relativos al tiempo total consumido dentro del servicio durante la ejecución completa de las aplicaciones. A cada una de ellas le corresponden tres barras: la primera indica el tiempo que consumiría el servicio especializado en caso de no existir reuso alguno (hit ratio HR=0%), la segunda indica el tiempo consumido por el servicio especializado durante los experimentos realizados (cache de resultados de 4095 entradas y mapeo directo), y la tercera indica el tiempo que se obtendría si se consiguiera un reuso del 100%. Las etiquetas del eje X indican, para cada aplicación, la función especializada y el reuso obtenido con 4095 entradas en la cache.

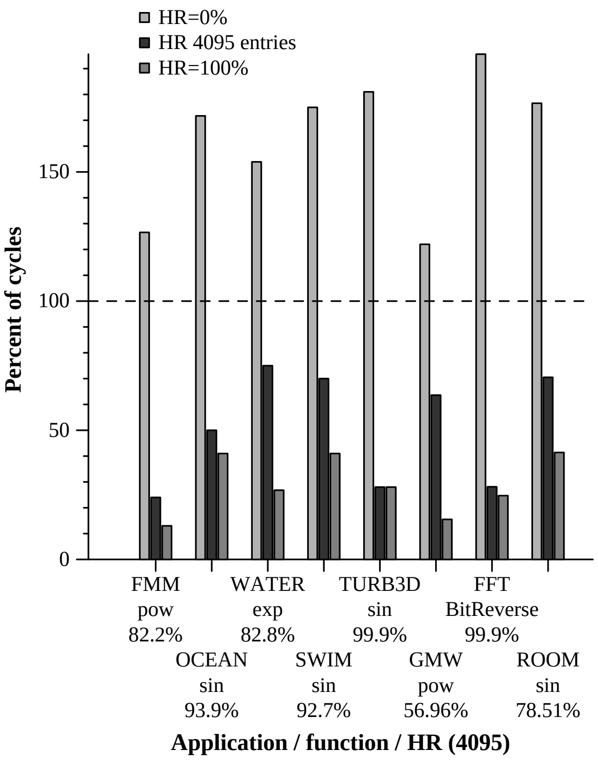

Fig. 3. Efecto de la especialización sobre los servicios

Puede observarse que el reuso es siempre suficientemente alto como para compensar la extensión y que, en ocasiones, la mejora llega a estar muy cerca del límite alcanzable.

Como es lógico, el efecto en el tiempo de ejecución de la aplicación se corresponde con el peso del servicio especializado en el tiempo total. En estos ejemplos, la función con más peso en la ejecución es `BitReverse()` de FFT (22.6%), y la mejora en tiempo de ejecución asociada es del 7.4%. En el resto de benchmarks, el peso del servicio especializado no supera el 10%, y las mejoras asociadas se mueven entre el 0.3% y el 8%.

B. Ampliación de la funcionalidad

Otra de las posibilidades que hemos mencionado consiste en utilizar las extensiones para introducir nuevas funcionalidades en el entorno de forma transparente. El ejemplo que presentamos en este apartado utiliza las DITools con la finalidad de monitorizar programas paralelos (ver [6] para más información).

Este tipo de monitorización es posible debido a que la aplicación paralela necesita servicios del entorno en puntos clave de su ejecución, ya sea para abrir paralelismo, para repartir el trabajo, o para sincronizar los flujos. La figura 4 muestra cómo se puede reconstruir el estado de cada uno de los flujos de la aplicación en base a la utilización de los servicios del entorno (en este caso el *runtime* paralelo). En esta figura, los nodos representan los estados que interesa conocer (*idle*, *run*, *scheduling*, y *blocked/synchronizing*) y las aristas representan transiciones provocadas por llamadas o retornos de servicios del entorno (las llamadas van precedidas por el signo +, y los retornos por el signo -).

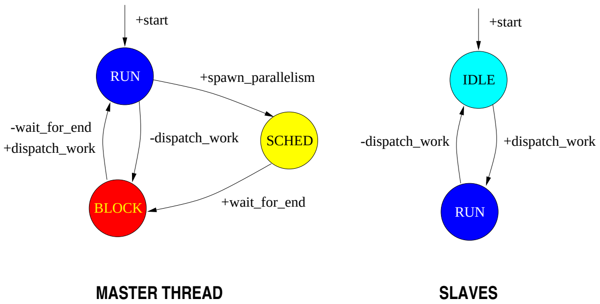

Fig. 4. Estados y transiciones detectables

Con esta información se puede construir el DIBackend apropiado para generar una traza temporal del estado de cada uno de los flujos de la aplicación. En este caso, usamos el estilo *callback handler* para extender cada uno de los servicios implicados con el siguiente código:

```
void pre_callback_handler()
{
  register_event(processor, timestamp, event, START);
}

void post_callback_handler()
{
  register_event(processor, timestamp, event, STOP);
}
```

Una vez generada, esta traza puede analizarse con las herramientas apropiadas. En la figura 5 se muestra un ejemplo de visualización con Paraver [7] del resultado de la monitorización de una aplicación paralela (derivada del benchmark TURB3D de los SPEC-95) con 8 procesadores. Todos los resultados que siguen han sido obtenidos en una Silicon Graphics Origin 2000 con el sistema operativo IRIX [10].

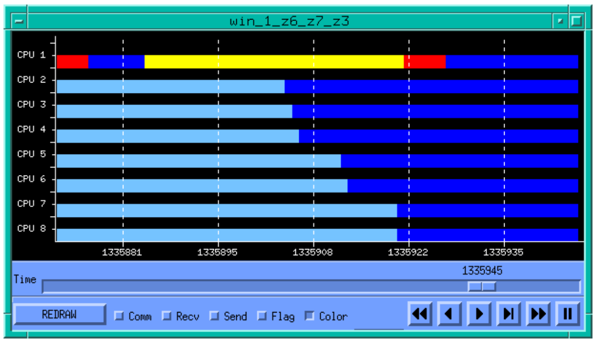

Fig. 5. Detalle de la ejecución (*fork*)

La figura muestra un intervalo de la ejecución que se corresponde con el principio de una fase paralela. El intervalo va del microsegundo 1.335.881 al microsegundo 1.335.935 aproximadamente. Hay una barra horizontal por procesador y los diferentes colores representan los diferentes estados. Si nos fijamos en la primera fila, podemos ver que el flujo empieza *blocked*, hay una transición a *run* cercana al microsegundo 1.335.881, para luego pasar a *scheduling*, brevemente a *blocked* y finalmente a *run*. El estudio de este intervalo de ejecución permite al diseñador del servicio (en este caso la librería de soporte al paralelismo) ver qué sucede realmente cuando el flujo principal distribuye el trabajo a los flujos esclavos. Puede observar el retardo en la iniciación de los flujos y determinar el coste de esta operación.

A más alto nivel, este tipo de monitorización permite obtener información útil para el programador de la aplicación, como el tiempo efectivo de cómputo de cada flujo, o la escalabilidad de su aplicación. En la figura 6 se muestra la ejecución completa del benchmark acompañada de un gráfico (izquierda) donde se muestra la proporción de tiempo en estado *run* de cada flujo sobre el tiempo total. Puede observarse que, con este número de procesadores, hay un buen balanceo de la carga.

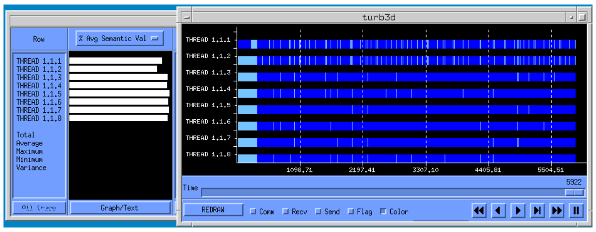

Fig. 6. Ejecución completa de la aplicación

V. Aplicaciones a HPC

Creemos interesante dedicar una sección de este trabajo a las aplicaciones de la extensibilidad mediante interposición a la computación de altas prestaciones (High Performance Computing), bien porque permiten obtener información de alto nivel útil para comprender y mejorar el rendimiento de las aplicaciones, bien porque facilitan la intervención en la ejecución mediante la introducción de cambios incrementales en los servicios.

A. Estudio de la ejecución de aplicaciones

Mejorar el rendimiento de programas (paralelos o no) de los que se posee poca información es uno de los problemas con los que a menudo se enfrenta la HPC.

Muchas veces resulta de gran ayuda un análisis de alto nivel de la ejecución, que nos permita comprender la estructura de la aplicación a nivel de módulos, de llamadas a funciones, y de tiempos.

Con los mecanismos y las herramientas que hemos descrito, resulta remarcablemente sencillo obtener dicho perfil de ejecución, así como refinarlo sucesivamente, eliminando la información irrelevante a medida que aumenta la comprensión sobre el funcionamiento del programa.

En la sección anterior, hemos descrito un experimento en el cual únicamente interesaba estudiar un aspecto muy específico de la ejecución: su interacción con el *runtime* paralelo. Resulta sencillo extender el ámbito del análisis a la aplicación completa para obtener un *perfil* global de su actividad, en el cual se reflejen todas las llamadas a funciones. En base a un análisis de este perfil inicial, puede centrarse la atención en el subconjunto de interfaces que más interese.

La siguiente traza muestra un fragmento de la ejecución del flujo principal del benchmark BT de NAS:

```
...
2623262   +178 x_solve_
2623486     +175 lhsx_
2638509     -175
2638738     +186 x_solve_cell_
2673173     -186
2673412     +185 x_backsubstitute_
2674186     -185
2674361   -178
2674493   +191 y_solve_
2674630     +176 lhsy_
2678248     -176
2678449     +199 y_solve_cell_
2741176     -199
2741403     +198 y_backsubstitute_
2741949     -198
2742171   -191
2742313   +200 z_solve_
2742449     +177 lhsz_
2756275     -177
2756484     +208 z_solve_cell_
2796169     -208
2796556     +207 z_backsubstitute_
2797778     -207
2797960   -200
2798100   +209 add_
2798667   -209
2798810   +109 initialize_
2827723   -109
2828114   +173 copy_faces_
2828257     +174 compute_rhs_
2834257     -174
2834458   -173
2834599   +178 x_solve_
...
```

En esta traza aparece a la izquierda el tiempo absoluto en nanosegundos, el signo (+/-) indica si se trata de una llamada o un retorno de función, la indentación refleja el nivel de anidamiento, el número es un identificador del evento, y va seguido del nombre de la función. El fragmento contiene la ejecución completa de varias rutinas, entre ellas `x_solve_`, `y_solve_`, y `z_solve_`. Para obtenerla, se ha empezado con el perfil completo (todas las llamadas y retornos de funciones). Este perfil se ha ido refinando eliminando los eventos que no aportaban información útil, hasta que ha aparecido esta estructura. En ningun momento ha sido necesario disponer del código fuente de BT ni de las librerías que utiliza.

B. Intervención en la ejecución

La facilidad con la que se pueden añadir extensiones mediante las herramientas propuestas permite evaluar de forma rápida el efecto de introducir cambios en el entorno de ejecución, una necesidad frecuente en la investigación que relaciona la HPC con los sistemas operativos y las aplicaciones.

Actualmente la herramienta se está utilizando para tomar medidas relativas a entrada/salida, con la intención de mejorar las políticas de entrada/salida para las aplicaciones paralelas. También se ha utilizado para experimentar con políticas de planificación de procesadores del sistema.

Ilustramos la última aplicación citada con un ejemplo. La figura 7 se ha obtenido después de haber modificado la política de planificación de procesadores del sistema (en este caso IRIX) mediante interposición (véase [1]).

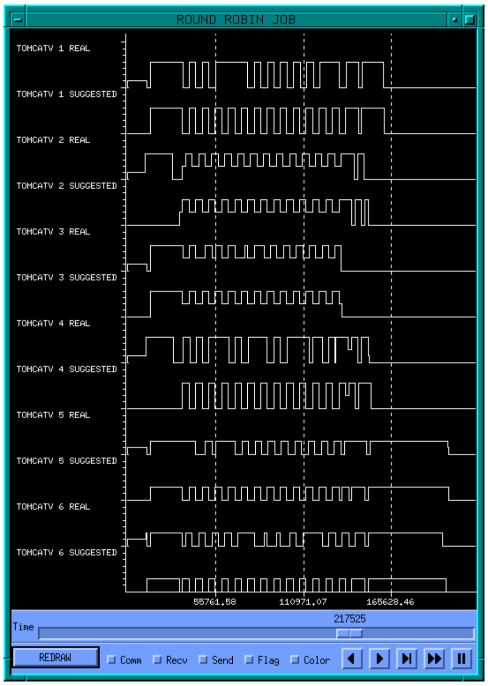

Fig. 7. Planificación con Round-Robin

Este experimento se repitió con varias políticas de planificación con el objeto de evaluar la viabilidad de un gestor de procesadores centralizado a nivel de usuario. En la figura se muestra el efecto de una política Round-Robin a nivel global, que reasigna los procesadores entre las aplicaciones. Cada una de las aplicaciones dispone de un flujo que comunica las peticiones. Se ha sometido al sistema a una carga de seis aplicaciones TOMCATV (perteneciente al SPEC-95), cada una de ellas con diferentes requerimientos en cuanto a procesadores. A cada aplicación le corresponden dos líneas: la primera de ellas indica el número de procesadores que el programa está utilizando en cada instante, y la segunda refleja el número de procesadores que el sistema le sugiere utilizar. Puede verse como, en líneas generales, los programas siguen las indicaciones de la nueva política, ajustando la cantidad de procesadores que utilizan al número asignado.

VI. Sumario

En este artículo hemos presentado el estado de nuestro trabajo en extensibilidad del entorno de ejecución, las herramientas que hemos desarrollado (DITools), ejemplos de su uso y algunas de sus aplicaciones prácticas a la HPC.

Las DITools son un conjunto de herramientas que permiten, de una manera cómoda y rápida, extender programas mediante interposición sin necesidad de generar nuevas versiones del ejecutable ni de las librerías. Hemos mostrado cómo pueden ser usadas para la especialización de servicios y para la introducción de nuevas funcionalidades de forma transparente, haciendo énfasis en el hecho de que sus aplicaciones a la HPC son directas.

Actualmente, estas herramientas se están utilizando en este campo de investigación gracias a que permiten una aproximación razonablemente sencilla a la obtención de información de alto nivel de la ejecución de las aplicaciones en condiciones reales, y a la intervención directa en ésta ejecución mediante extensiones incrementales del entorno.

VII. Agradecimientos

Queremos agradecer a los miembros del proyecto Esprit Nanos [5] la oportunidad de experimentar conjuntamente aplicando nuestras herramientas a su entorno y sus aplicaciones, a Sergi Aldomà y a los miembros del *Staff* del CEPBA la ayuda y el soporte que nos han dado, y a José González por sus comentarios en versiones preliminares de este artículo.

References

[1] Sergi Aldomà *Mecanismes en el Kernel d'un Sistema Operatiu pel Control de l'Execució d'Aplicacions Paral.leles* M.S.Thesis, Departament d'Arquitectura de Computadors - UPC, February 1999, written in Catalan.

[2] Ernest Artiaga, Nacho Navarro, Xavier Martorell, Yolanda Becerra, Marisa Gil, Albert Serra *Experiences on the Implementation of PARMACS Macros Using Different Multiprocessor Operating System Interfaces* VIII Jornadas de Paralelismo, 1997

[3] D.Ghormley, S.Rodrigues, D.Petrou, T.Anderson *Interposition as an Operating System Extension Mechanism* Technical Report TR-CSD-96-920, University of California at Berkeley, Feb. 1998

[4] M.Jones *Interposition Agents: Transparently Interposing User Code at the System Interface* In Proceedings of the 14th SOSP, Dec. 1993

[5] Nanos Consortium *Nano-Threads: Programming Model Specification* Nanos project deliverable, http://www.ac.upc.es/nanos, July 1997

[6] José Oliver, Xavier Martorell, Albert Serra, Marc Tallada, Eduard Ayguadé, Nacho Navarro, Jesús Labarta *Performance Analysis for OpenMP Applications* Submitted to SC'99

[7] V.Pillet, J.Labarta, T.Cortés, S.Girona *PARAVER: A Tool to Visualize and Analyse Parallel Code* WoTUG-19, pp. 17-31, Manchester, April 1995.

[8] Albert Serra, Nacho Navarro *Dynamically–linking extensions and the Memoization Experience* Technical Report UPC-DAC-1999-17

[9] Albert Serra, Nacho Navarro *Extending the Execution Environment with* DITools Technical Report UPC-DAC-1999-26

[10] Silicon Graphics Computer Systems (SGI) *Origin2000 and Onyx2 Performance Tuning and Optimization Guide* Doc. num. 007-3430-002, http://techpubs.sgi.com, 1998