Servet: una suite de benchmarks para el soporte del autotuning en clusters de sistemas multinúcleo

Jorge González-Domínguez, Guillermo L. Taboada, Basilio B. Fraguela, María J. Martín, Juan Touriño

Grupo de Arquitectura de Computadores, Universidad de A Coruña

Email: *{jgonzalezd, taboada, basilio.fraguela, mariam, juan}@udc.es*

Resumen

El presente artículo describe Servet, una suite de benchmarks centrada en detectar un conjunto de parámetros hardware con una gran influencia en el rendimiento de las aplicaciones ejecutadas en clusters de sistemas multinúcleo, los cuales presentan una complejidad creciente en cuanto a su estructura y configuración. Estos benchmarks detectan la jerarquía caché, los anchos de banda en el acceso concurrente a memoria y las latencias de comunicaciones entre los núcleos. Servet se ha probado en varios sistemas obteniendo estimaciones muy precisas de los parámetros de la máquina, siendo de gran utilidad para el soporte del autotuning.

1. Introducción

En la actualidad existe una creciente tendencia a desarrollar códigos que tengan en cuenta el sistema en el que están siendo ejecutados para realizar optimizaciones de forma automática. Entre las diferentes arquitecturas paralelas, la optimización de aplicaciones en los clusters de sistemas multinúcleo presenta un importante desafío ya que poseen una arquitectura de memoria híbrida (distribuida/compartida) con latencias de comunicaciones no uniformes. Además, accesos concurrentes a la misma caché o al mismo módulo de memoria desde distintos núcleos pueden desembocar en una pérdida de ancho de banda a memoria.

Actualmente existen dos líneas para mejorar el rendimiento de las aplicaciones paralelas en estos sistemas. Por un lado implementar técnicas que tengan en cuenta las características de los sistemas [1, 6], tales como minimizar el número de mensajes a través de las redes de interconexión o usar bloques de datos que quepan en las cachés para evitar los fallos caché. La segunda opción consiste en asignar los procesos a núcleos específicos para mejorar el rendimiento sin modificar los códigos [2, 3]. En ambas aproximaciones es necesario el conocimiento de la topología de la máquina y de algunos parámetros hardware.

El uso de benchmarks es el único método general y portable para averiguar las características del hardware sin preocuparse por el sistema operativo o los privilegios que posee el usuario. Además, esta aproximación proporciona resultados experimentales acerca del rendimiento de los sistemas, con una estimación más fiable que obteniéndolos de las especificaciones de la máquina.

Este artículo presenta Servet, un conjunto de benchmarks portables para obtener los parámetros hardware de los clusters de sistemas multinúcleo y, por tanto, dar soporte a la optimización automática de los códigos paralelos en esas arquitecturas. Los parámetros estimados son la jerarquía caché, las sobrecargas en el acceso a memoria y las capas de comunicaciones con diferentes latencias y anchos de banda. Se ha comprobado la precisión de las estimacionas de Servet en varios sistemas con diferentes arquitecturas.

1.1. Trabajo relacionado

La extracción automática de los parámetros de sistemas multinúcleo ha sido tratada con an-

terioridad mediante distintas aproximaciones. Servet ha superado ciertas carencias como la falta de portabilidad o la ausencia del estudio de ciertos parámetros hardware importantes, tales como el análisis de los distintos niveles de sobrecarga de acceso a memoria o de la jerarquía de comunicaciones.

Un estudio más exhaustivo de los trabajos relacionados se encuentra recogido en [4].

2. La suite de benchmarks

En esta sección se describen los benchmarks implementados en Servet para determinar los tamaños de las cachés, la topología de las cachés compartidas, las sobrecargas en el acceso a memoria y el rendimiento de las comunicaciones.

2.1. Estimación de los tamaños caché

La idea principal para estimar los tamaños de las cachés sigue la aproximación de Saavedra y Smith [5]: calcular el número de ciclos necesarios para recorrer arrays de diferentes tamaños. Los accesos a los elementos de los arrays se hacen en intervalos de 1KB, que son lo suficientemente grandes para evitar influencias de los prebuscadores hardware actuales, que solo trabajan con intervalos de hasta 512 bytes. Además, este intervalo es mayor que cualquier tamaño de línea existente y, al mismo tiempo, es divisor del tamaño de cualquier caché.

Sin embargo, esta aproximación presenta algunos inconvenientes: los resultados podrían estar distorsionados por las optimizaciones de los compiladores más agresivos y deben ser interpretados manualmente para determinar el tamaño de las diferentes cachés. Servet mejora este algoritmo usando siempre los valores de un array como intervalo de acceso, evitando así los efectos de las optimizaciones de los compiladores, y proporcionando directamente los tamaños de las cachés.

Siguiendo esta idea, el algoritmo de la Figura 1 (*mcalibrator*) proporciona como salidas dos arrays `S` y `C`, de longitud `n`, que contienen, respectivamente, los tamaños de los arrays recorridos y el número medio de ciclos para acceder a un elemento de esos arrays.

```
aux = 0 //Variable auxiliar
i = MIN_CACHE
n = 0 //Número de tamaños caché probados
while i≤MAX_CACHE do
    S[n] = i //Tamaño del array recorrido
    size = Elementos en S[n] bytes
    for j=0;j<size;j=j+1 do
        //Todos guardan el intervalo
        A[j] = Elementos en 1KB
    end
    //Accesos al array
    for j=0;j<size;j=j+A[j] do
        aux = aux+size
    end
    C[n] = Ciclos usados en bucle anterior
    n = n+1
    if i<2MB then
        i = i*2
    else
        i = i+1MB
    end
end
```

Figura 1: *mcalibrator*

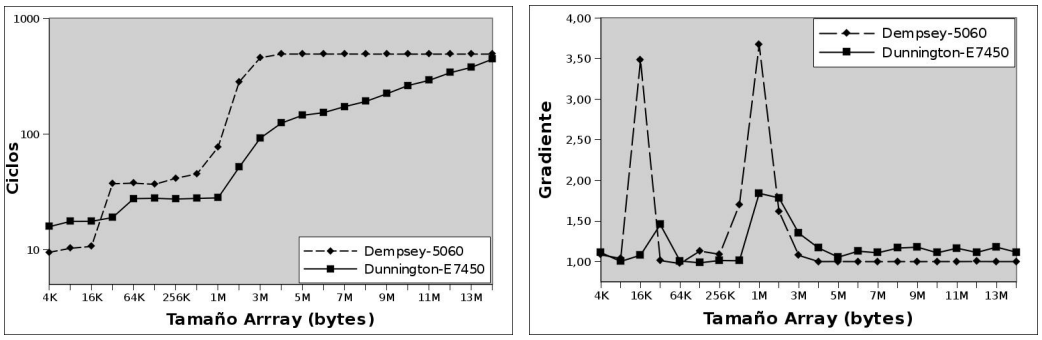
La Figura 2(a) muestra los ciclos medios necesarios para acceder a los elementos de arrays de diferentes tamaños en dos máquinas Intel Xeon (*Dempsey*-5060 y *Dunnington*-E7450), mientras que la Figura 2(b) muestra los gradientes, es decir, $C[k+1]/C[k], 0 \leq k < n$.

Caché L1

Recorrer un array con todos sus elementos en caché es más rápido que recorrer uno que no cabe en caché, debido a que en el último caso se originan más fallos caché. Por tanto, un incremento en el número medio de ciclos indica que ese tamaño excede el de un determinado nivel de caché. Cambios agudos en el gráfico de los ciclos se corresponden con picos en el gráfico que muestra los gradientes. El tamaño de la caché L1 se determina tomando el primer pico de los gradientes: 16KB para *Dempsey* y 32KB para *Dunnington*.

Otros niveles caché

La aproximación para estimar el tamaño de las cachés L1 no es aplicable a otros niveles,



(a) Número medio de ciclos para acceder a un elemento de un array

(b) Gradiente del número de ciclos

Figura 2: Resultados del *mcalibrator*

ya que las cachés L1 suelen estar indexadas virtualmente, pero niveles más bajos se suelen indexar físicamente. Esto origina un problema cuando el tamaño de la caché es mayor que el de una página virtual porque la contigüidad en memoria virtual no implica adyacencia en memoria física, originando fallos caché con arrays más pequeños que la caché. De este modo, la búsqueda de picos en la función gradiente no es suficiente para estimar correctamente el tamaño de dichos niveles caché. Un ejemplo es la máquina *Dempsey*, con gradientes elevados en el rango [512KB,2MB]. Aunque algunos sistemas operativos resuelven este problema aplicando la técnica del coloreado de página, otros como LINUX no lo hacen.

El benchmark de Servet supera este problema de forma portable con una aproximación probabilística. Dado que un sistema operativo puede asignar una página virtual a cualquier página física, no se puede asumir que un determinado conjunto de una caché indexada físicamente se corresponda con una página virtual. Suponiendo que se dispone de un sistema con tamaño de página PS y una caché indexada físicamente de tamaño CS con asociatividad K, cada vía de la caché se puede dividir en $CS/(K * PS)$ conjuntos de página, es decir, grupos de conjuntos caché que pueden recibir datos de la misma página. Por tanto, si la probabilidad de que cierta página virtual sea asignada a un conjunto de página concreto es uniforme, el número de páginas X por conjunto de página pertenece a una binomial $B(NP, (K*PS)/CS)$, donde NP es el número de páginas involucradas en el acceso [1]. Puesto que cada conjunto puede almacenar hasta K páginas sin conflictos, la probabilidad de $P(X > K)$ en la binomial anterior es el ratio de fallos teórico.

De este modo, empleando las salidas de *mcalibrator* (Figura 1), el algoritmo de la Figura 3 aplica el razonamiento previo para estimar los tamaños de las cachés L2 y L3. Este algoritmo probabilístico comienza calculando el número de páginas y el ratio de fallo caché por cada experimento del *mcalibrator*. Tras esto, calcula las divergencias entre los ratios experimentales y los teóricos según la distribución binomial. La estimación será el tamaño caché con menor divergencia.

La precisión de este algoritmo es mayor que buscando picos en las salidas del *mcalibrator*. Por ejemplo, estudiando el segundo pico, en la máquina *Dempsey* se estimaría el tamaño de la L2 erróneamente como 1MB. Este último algoritmo, analizando como posibles tamaños caché el rango [256KB,4MB], obtiene como salida el tamaño correcto de 2MB.

Este algoritmo también es capaz de estimar correctamente el tamaño de una caché si varios gradientes consecutivos son mayores que 1. Es-

```
hit_time=MIN(C)
miss_overhead=MAX(C)-MIN(C)
for i=0;i<n;i=i+1 do
    //Ratio de fallo experimental
    MR[i]=(C[i]-hit_time)/miss_overhead
    NP[i]=S[i]/PS //Número de páginas
end
foreach posible CS y K do
    div[CS][K] = 0
    for i=0;i<n;i=i+1 do
        div[CS][K] += | MR[i]-P(X>K) |,
        X∈ B(NP[i], (K*PS)/CS)
    end
end
Result: CS con menor div
```

Figura 3: Algoritmo probabilístico para estimar el tamaño de las cachés indexadas físicamente

ta situación ocurre en la máquina *Dunnington* (ver Figura 2): analizando con este algoritmo el rango [3MB,14MB] la salida estima una caché de 12MB, el tamaño real de la caché L3.

En resumen, los tamaños de las cachés L1 (virtualmente indexadas) se estiman siempre como la posición del primer pico de los gradientes. Sin embargo, para los siguientes niveles hay dos posibilidades. Si existe un pico claramente localizado en un único tamaño de array significa que el sistema operativo aplicó el coloreado de páginas y, como el comportamiento de la caché es similar al de una virtualmente indexada, la posición del pico también sirve para estimar su tamaño. En el caso de que el pico tenga gradientes mayores que 1 para varios tamaños de array se emplea el algoritmo probabilístico. De esta forma la estimación es correcta independientemente de si el sistema operativo aplica el coloreado de páginas o no.

2.2. Determinación de cachés compartidas

El conocimiento de qué núcleos comparten una determinada caché puede ser muy útil para acelerar los accesos a memoria. La Figura 4 muestra el algoritmo para detectar la topología de las cachés compartidas. Las entradas son el número de niveles caché l y un array CS con el tamaño de cada caché. Por cada nivel i, se empieza llamando al *mcalibrator* con un array mayor que $CS[i]/2$ y guardando el resultado como referencia. Tras esto, se invoca al *mcalibrator* simultáneamente desde dos procesos en núcleos distintos accediendo a un array del mismo tamaño que el usado para obtener la referencia. El tamaño escogido provoca que dos arrays no quepan en la misma caché, originando reemplazos adicionales en la caché cuando ésta es compartida e incrementando el número de ciclos. La salida es un array de listas P_{sc} (una lista por cada nivel caché) con los pares de núcleos que presentan un número de ciclos significativamente mayor que la referencia (ratio $>$ 1.5), lo que significa que comparten esa caché.

```
Data: l, //número de niveles caché
CS[0..l-1] //tamaño caché por nivel
for i=0;i<l;i=i+1 do
    P_sc[i] = Lista vacía
    ref = Ciclos del mcalibrator con un núcleo
    y un array de tamaño (2/3) * CS[i]
    foreach posible par de núcleos do
        c = Ciclos del mcalibrator con los dos
        nucleos y arrays de tamaño
        (2/3) * CS[i] ratio = c/ref
        if ratio>1.5 then
            Añadir el par a P_sc[i]
        end
    end
end
Result: P_sc[0..l-1]
```

Figura 4: Detección de cachés compartidas

2.3. Caracterización de sobrecargas en el acceso a memoria

En sistemas con memoria compartida los accesos concurrentes a memoria pueden convertirse en un cuello de botella. Servet proporciona un benchmark para detectar las sobrecargas en el acceso a memoria que consiste en comparar el ancho de banda a memoria cuando accede un único núcleo de forma aislada con el que obtienen los núcleos al acceder por parejas. Para calcular el ancho de banda se copian todos los elementos almacenados en un array a otro (estos arrays no pueden caber en ninguna caché).

Pueden existir sobrecargas de distinta magnitud y Servet proporciona información acerca de qué pares de núcleos sufren cada una de las sobrecargas. Para ello se trabaja con dos arrays como se muestra en el algoritmo de la Figura 5: BW, con los distintos anchos de banda menores que la referencia, y P_m, con las listas de pares de núcleos relacionados con cada sobrecarga ($P_m[i]$ es la lista de pares de núcleos con un ancho de banda concurrente similar a $BW[i]$).

```
n = 0 //Número de sobrecargas
ref = Ancho de banda a memoria con un
único núcleo
foreach posible par de núcleos do
    b = Ancho de banda a memoria
    accediendo ambos núcleos
    if b<ref then
        if b es similar a algún BW[i],
        0≤i<n then
            Añadir el par a P_m[i]
        else
            BW[n] = b
            P_m[n] = Lista vacía
            Añadir el par a P_m[n]
            n = n+1
        end
    end
end
Result: n, BW[0..n-1], P_m[0..n-1]
```

Figura 5: Algoritmo para caracterizar las sobrecargas en el acceso a memoria

Una vez obtenidas las sobrecargas, usando los arrays BW y P_m, la escalabilidad del ancho de banda efectivo se puede obtener usando tan solo los pares de núcleos de un grupo representativo de cada sobrecarga.

2.4. Determinación de la jerarquía de las comunicaciones

La caracterización de las comunicaciones proporcionada por Servet se divide en dos partes. En primer lugar, se determinan las capas de comunicación (conjuntos de pares de núcleos cuyos costes de comunicación son muy similares). Una vez detectadas, se muestra información acerca del comportamiento de cada capa según el tamaño del mensaje a enviar.

Para agrupar los núcleos de acuerdo a los costes de sus comunicaciones se ha implementado el benchmark de la Figura 6. La implementación de referencia usa MPI. Este algoritmo compara las latencias al enviar un mensaje entre cada par de núcleos, almacenando el valor de las diferentes latencias en el array L. En la posición i de P_l se guarda una lista con los pares con los que se ha obtenido la latencia $L[i]$. Se pueden emplear mensajes de distintos tamaños para este cometido pero, en el caso de Servet, se envían mensajes de igual tamaño que el de la caché L1, lo que permite detectar las diferencias en las latencias por compartir cachés de otros niveles.

```
n = 0 //Número de latencias diferentes
foreach posible par de núcleos do
    l = Latencia de enviar un mensaje entre
    los dos núcleos
    if l es similar a algún L[i], 0≤i<l
    then
        Añadir el par a P_l[i]
    else
        L[n] = l
        P_l[n] = Lista vacía
        Añadir el par a P_l[n]
        n = n+1
    end
end
Result: n, L[0..n-1], P_l[0..n-1]
```

Figura 6: Detección de capas de comunicaciones

Tras establecer las capas, se guardan los resultados de un micro-benchmark de comunicaciones punto a punto para tamaños de mensaje representativos. Estos resultados se obtienen para un par de núcleos representativo de cada capa, relacionando el rendimiento de los demás pares del grupo con el de dicho par de referencia.

3. Evaluación experimental

La evaluación general de Servet se ha realizado en dos entornos distintos. Primero en un sistema con 4 procesadores *Dunnington* Intel Xeon E7450 de seis núcleos cada uno (2,40 GHz).

Todos los núcleos de cada procesador comparten una caché L3 de 12MB mientras que las L2 de 3MB están compartidas por parejas de núcleos. Las cachés L1 de 16KB son independientes. La implementación de MPI del sistema es la MPICH2 versión 1.1.1.

El segundo entorno es el supercomputador Finis Terrae del Centro de Supercomputación de Galicia (CESGA). Consta de 142 nodos HP RX7640, cada uno de ellos con 8 procesadores de doble núcleo (1,60 GHz), los 16 núcleos están distribuidos en dos celdas (cada una de ellas con 4 procesadores y 8 núcleos). Cada celda posee su propia memoria de 64GB, así que todos los núcleos en el mismo nodo comparten lógicamente 128GB de memoria. Los accesos a memoria se realizan a través de buses compartidos por pares de procesadores (4 núcleos). Cada núcleo posee tres cachés individuales: L1 de 16 KB, L2 de 256KB y L3 de 9MB. Los nodos se conectan vía red InfiniBand (20 Gbps). Se emplea el compilador HP MPI 2.2.5.1.

3.1. Estimación de los tamaños caché

Con el objeto de validar la estimación del tamaño de los distintos niveles de caché en un número amplio de máquinas, se solicitó a los alumnos de la asignatura de Estructura de Computadores II de la Universidad de A Coruña que ejecutasen esta parte de Servet en sus máquinas. Así, se utilizaron 70 máquinas distintas, englobando un total de 147 cachés de distinto nivel. Servet ha proporcionado la estimación correcta en 140 de esas cachés, alcanzando más de un 95 % de acierto. En los restantes fallos las diferencias entre los tamaños reales y las estimaciones fueron menores.

3.2. Determinación de cachés compartidas

La Figura 7(a) muestra los resultados de la detección de las cachés compartidas (Sección 2.2) en el sistema con los 4 procesadores *Dunnington*, mientras la Figura 7(b) muestra los resultados para un nodo del *Finis Terrae*. Sólo se enseñan los resultados para los pares con el núcleo 0 por cuestiones de claridad. La medida del rendimiento es el ratio de la sobrecarga por accesos concurrentes a la caché presentado en el benchmark (ver *ratio* en la Figura 4).

En *Dunnington*, observando los ratios mayores que 1.5, el núcleo 0 comparte la caché L2 con el núcleo 12 y la L3 es compartida por todos los núcleos en el mismo procesador: {0,1,2,12,13,14}. En el *Finis Terrae*, con todos los ratios menores de 1.5, todas las cachés son individuales.

3.3. Caracterización de sobrecargas en el acceso a memoria

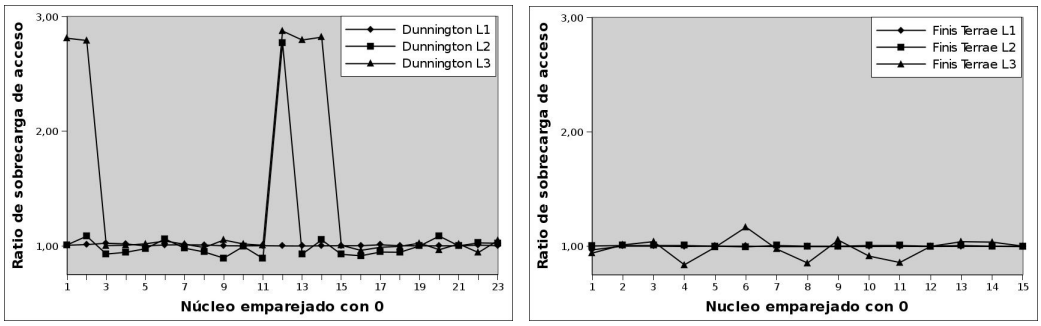
La Figura 8(a) muestra el ancho de banda con dos núcleos accediendo concurrentemente a memoria tanto en el sistema *Dunnington* como en el *Finis Terrae*. De nuevo, por claridad, solo se muestran las parejas que incluyen el núcleo 0. Además, el primer valor, etiquetado como *ref* en el eje x, es el ancho de banda al acceder a memoria un único núcleo.

En el sistema *Dunnington* se observa sólo un tipo de sobrecarga cuando cualquier par de núcleos accede a memoria simultáneamente. Sin embargo, la sobrecarga varía en el *Finis Terrae* dependiendo del par de núcleos estudiado. El menor ancho de banda aparece cuando el núcleo 0 accede al mismo tiempo que los núcleos 1, 2 o 3 debido a que estos núcleos pertenecen a procesadores que comparten el mismo bus a memoria. Un ancho de banda mayor se obtiene con los núcleos del 4 al 7, pero éste es aún un 25 % menor que el valor de referencia porque estos núcleos se encuentran en la misma celda que el 0, compartiendo memoria. Por último, si se analizan núcleos de distintos nodos, no se observa sobrecarga alguna.

La Figura 8(b) muestra el ancho de banda efectivo de acceso a memoria cuando varios núcleos acceden simultáneamente. Sólo se analizan los grupos de núcleos que muestran sobrecarga en la gráfica de la izquierda.

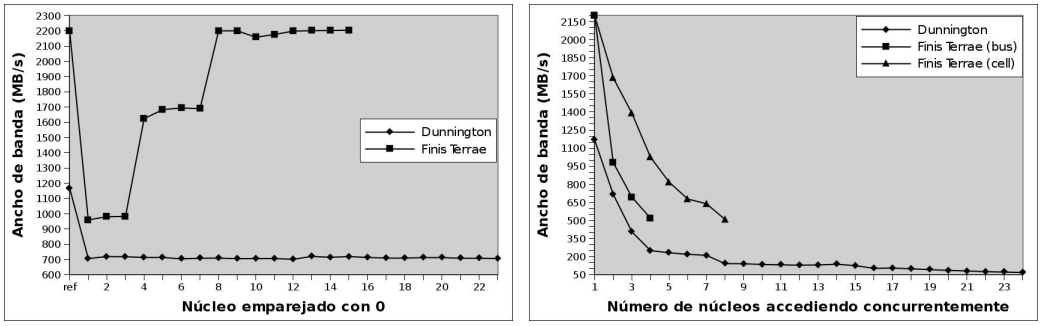
3.4. Determinación de la jerarquía de las comunicaciones

La Figura 9(a) muestra la latencia de los mensajes MPI entre el núcleo 0 y los restantes núcleos. En el sistema *Dunnington*, las comunicaciones entre núcleos del mismo procesador pre-



(a) *Dunnington* (b) *Finis Terrae*

Figura 7: Resultados del benchmark de detección de cachés compartidas



(a) Ancho de banda con dos accesos simultáneos (b) Ancho de banda con varios accesos simultáneos

Figura 8: Resultados del benchmark de caracterización de sobrecargas en el acceso a memoria

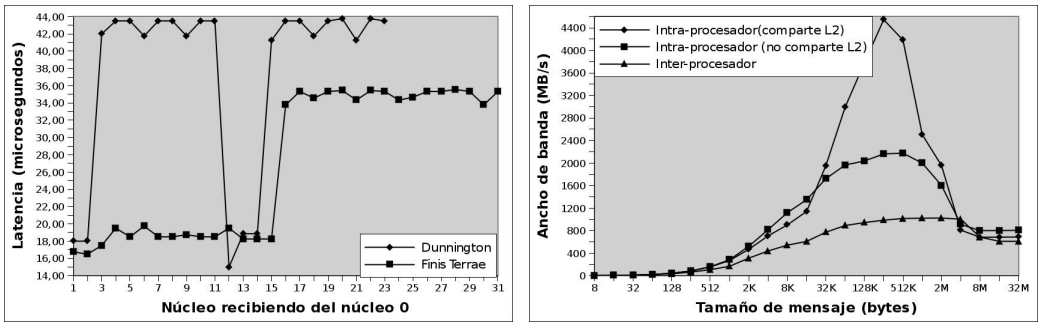
sentan las latencias más bajas, especialmente entre núcleos que comparten la caché L2 (núcleos 0 y 12). En el *Finis Terrae*, en este caso, se usan 2 nodos (32 núcleos), suficiente para caracterizar todos los diferentes costes de comunicaciones. Las comunicaciones intra-nodo (núcleos del 0 al 15) son casi el doble de rápidas que las inter-nodo (núcleos del 16 al 31), a través de InfiniBand.

La Figura 9(b) presenta el ancho de banda punto a punto para pares de núcleos representativos del sistema *Dunnington*, uno por cada capa detectada con el algoritmo de la Figura 6. Estos resultados permiten estimar el ancho de banda de comunicaciones para cualquier mensaje, teniendo en cuenta tanto su tamaño como la capa de comunicación utilizada.

4. Conclusiones

En este artículo se ha presentado Servet, una herramienta que obtiene, mediante benchmarks, los parámetros hardware necesarios para la optimización automática de las aplicaciones en los clusters de sistemas multinúcleo. Las características reveladas por Servet son la topología y el tamaño de las cachés, las sobrecargas en los accesos a memoria y la jerarquía de comunicaciones. Las pruebas realizadas en diferentes sistemas han puesto de manifiesto que la herramienta estima de forma precisa estos parámetros.

Son numerosas las técnicas de optimización que pueden aprovecharse del conocimiento de estos parámetros hardware. Para empezar, el



(a) Latencia de envío de un mensaje (tamaño de L1) (b) Ancho de banda punto a punto (*Dunnington*)

Figura 9: Resultados del benchmark de comunicaciones

conocimiento de las sobrecargas permite asignar los procesos a núcleos específicos para evitar o minimizar los cuellos de botella de las aplicaciones. Por otro lado, la obtención automática de los tamaños de las cachés permite implementar una de las técnicas de optimización más utilizadas: la división interna de los cálculos para que trabajen con bloques de datos que quepan en caché. Además, las comunicaciones pueden optimizarse usando la información acerca de la jerarquía de comunicaciones. Todo ello permite explotar de forma más eficiente la localidad de los datos, disminuir la sobrecarga de las comunicaciones y minimizar la influencia de los cuellos de botella de las aplicaciones paralelas.

Servet se encuentra disponible bajo licencia GPL en `http://servet.des.udc.es`.

Agradecimientos

Este trabajo ha sido financiado por la Xunta de Galicia dentro del proyecto INCITE08PXIB105161PR y por el Ministerio de Educación y Ciencia tanto dentro del proyecto (TIN2007-67537-C03-02) como con la beca FPU AP2008-01578. Agradecemos a los alumnos de Estructura de Computadores II de la Universidad de A Coruña el habernos ayudado a probar la precisión de las estimaciones de Servet y al CESGA (Centro de Supercomputación de Galicia) por proporcionarnos acceso al supercomputador Finis Terrae.

Referencias

[1] B. B. Fraguela, Y. Voronenko and M. Püschel. Automatic Tuning of Discrete Fourier Transforms Driven by Analytical Modeling. En *18th Intl. Conf. on Parallel Architectures and Compilation Techniques (PACT'09)*, pp 271–280, Raleigh, NC, USA, 2009.

[2] G. Mercier and J. Clet-Ortega. Towards an Efficient Process Placement Policy for MPI Applications in Multicore Environments. En *Proc. 16th European PVM/MPI Users' Group Meeting (EuroPVM/MPI'09)*, LNCS v.5759, pp 104–115, Espoo, Finlandia, 2009.

[3] H. Chen, W. Chen, J. Huang, B. Robert and H. Kuhn. MPIPP: An Automatic Profile-guided Parallel Process Placement Toolset for SMP Clusters and Multiclusters. En *Proc. 20th Intl. Conf. on Supercomputing (ICS'06)*, pp 353–360, Cairns, Australia, 2006.

[4] J. González-Domínguez, G. L. Taboada, B. B. Fraguela, M. J. Martín and J. Touriño. Servet: A Benchmark Suite for Autotuning on Multicore Clusters. En *Proc. 24th Intl. Parallel and Distributed Processing Symp. (IPDPS'10)*, Atlanta, GE, USA, 2010.

[5] R. H. Saavedra and A. J. Smith. Measuring Cache and TLB Performance and their Effect on Benchmark Runtimes. *IEEE Trans. Computers*, 44(10):1223–1235, 1995.

[6] V. Tipparaju, J. Nieplocha and D. K. Panda. Fast Collective Operations Using Shared and Remote Memory Access Protocols on Clusters. En *Proc. 17th Intl. Parallel and Distributed Processing Symp. (IPDPS'03)*, pp 84–93, Nice, Francia, 2003.