Minimulticomputador de bajo coste

Carles Aliagas, Pere Millán, Carlos Molina
Dep. d'Enginyeria Informàtica i Matemàtiques
Universitat Rovira i Virgili
Tarragona
{carles.aliagas,pere.millan,carlos.molina}@urv.cat

Roc Meseguer
Dept. of Computer Architecture
Universitat Politècnica de Catalunya
Barcelona
meseguer@ac.upc.edu

Resumen

En la mayoría de los estudios de Grado en Ingeniería Informática hay asignaturas que abordan el tema de la supercomputación. Uno de sus objetivos es adquirir competencias en programación paralela. Para realizar ejercicios y prácticas se suelen usar estándares como OpenMP, MPI y CUDA. Para programar con dichos estándares se usan sistemas de elevado precio, lo que hace que el presupuesto disponible limite el número de procesadores. Por lo tanto, el acceso a un supercomputador con cientos de procesadores (que supone centenares de miles de euros) no parece estar justificado para realizar prácticas con los estudiantes. Sin embargo, y siguiendo la tendencia de usar muchos procesadores pero poco potentes basados en ARM, se puede construir un minimulticomputador de bajo coste por un precio equivalente a un servidor de memoria compartida. Este trabajo presenta un recurso docente basado en placas de HardKernel, que integran 64 placas Odroid y que mediante Gigabit-Ethernet permiten montar un servidor de programación MPI con 256 procesadores. Si bien se trata de un recurso de bajas prestaciones, es interesante el hecho de tener acceso a centenares de procesadores para poder hacer estudios de escalabilidad, manteniendo un buen compromiso entre prestaciones, precio y consumo.

Abstract

In most of Computer Science Degrees, there are subjects that address the topic of supercomputing. One of the objectives of these subjects is to acquire competences in parallel programming. To carry out exercises and practices, standards such as OpenMP, MPI and CUDA are often used. Unfortunately, the most suitable systems to deal with those standards are very expensive and most of the times the available budget limits the number of processors. Owning a supercomputer with hundreds of processors (that means hundreds of thousands of euros) does not seem to be justified in a teaching environment. However, assuming the trend of dealing with many low-power processors (based on ARM architectures), a low-cost minimulticomputer can be built for a price equivalent to a shared memory server. In this work, we present a teaching resource based on HardKernel boards, with 64 Odroid boards connected through Gigabit-Ethernet, to build a MPI server with 256 processors. Although it is a resource with a relatively low performance, the aim is to have access to hundreds of processors to be able to carry out scalability analysis and, above all, maintaining a good trade-off between performance, price and energy consumption.

Palabras clave

Supercomputación, paralelismo, HPC, MPI, ARM.

1. Motivación

Para poder adquirir las competencias de asignaturas de paralelismo y computación en los estudios de Grado en Ingeniería Informática, es necesario practicar y tener acceso a sistemas de computación paralelos y distribuidos. Se puede considerar que estándares como OpenMP, MPI y CUDA son los más habituales. Para poder utilizarlos se hace imprescindible tener acceso a hardware específico que permita realizar ejercicios y prácticas. Estos sistemas suelen tener precios elevados y, con frecuencia, el presupuesto limita el número de procesadores del sistema. Mientras que en el ámbito investigador los departamentos suelen tener acceso a sistemas de altas prestaciones, en el ámbito docente se hace difícil poder acceder a estos sistemas y se suele recurrir a equipos más modestos.

En la asignatura de cuarto curso del Grado en Ingeniería Informática de la Universitat Rovira i Virgili, objeto de este artículo, se utiliza un sistema de colas donde se organiza el acceso 24/7 a servidores dedicados. Así, tanto en horas lectivas como en horas personales, se tiene acceso a los servidores.

En nuestro caso, en las prácticas realizamos ejercicios para máquinas de memoria compartida (multiprocesador) y para máquinas de memoria distribuida (multicomputador). En el primer caso (memoria compartida) se programa con el estándar OpenMP y para realizar las ejecuciones se utiliza una máquina con dos procesadores Xeon E5-2660 a 2.2 GHz, disponiendo cada uno de ellos de 8 *cores* y con capacidad *multithread* (2 *threads*). De este modo, se pueden ejecutar hasta 32 procesos de manera simultánea. En el segundo caso (memoria distribuida) se estudia el modelo MPI de programación paralela y se utiliza para realizar las ejecuciones un sistema de 4 nodos con 2 procesadores Opteron 2210 a 1.8 GHz cada uno, disponiendo cada uno de ellos de 2 *cores*. De este modo, se pueden ejecutar hasta 16 procesos de manera simultánea.

Este tipo de entornos es el que se suele utilizar en ámbitos docentes de asignaturas que introducen a los alumnos en la programación paralela. Giménez [2] describe su experiencia en el curso "Introducción a la Programación Paralela" del Departamento de Informática y Sistemas de la Universidad de Murcia, que se centra en entornos y herramientas de programación paralela como OpenMP, MPI y CUDA. Para ello utilizan un *cluster* de seis nodos con un total de 64 núcleos y con tarjetas GPU y Xeon Phi para cada nodo (10 GPUs y 2 Xeon Phi). Santamaría *et al*. [4] en el curso de Arquitectura de Computadores de la titulación de Ingeniería de Telecomunicación de la Universidad de Jaén, utilizan ordenadores con tarjetas gráficas NVIDIA GeForce 8800 GT, así como el *toolkit* CUDA 2.3, pero la experiencia es aplicable a otras alternativas de paralelización (MPI, OpenMP).

En general, en entornos OpenMP y debido a que requieren sistemas de memoria compartida, es relativamente fácil tener acceso a sistemas de 16-32 procesadores por un precio que puede ir de 5.000 € a 10.000 €. Del mismo modo, un entorno CUDA es también relativamente asequible. Una tarjeta gráfica de gama alta se puede adquirir por un precio entre 1.000 € y 2.000 € y permite añadirla a un servidor para realizar los ejercicios de CUDA necesarios. Incluso podría ser el mismo servidor para OpenMP y CUDA. En cambio, para entornos de programación MPI, al ser sistemas de memoria distribuida, resulta interesante realizar estudios de escalabilidad (que es su principal ventaja respecto a OpenMP). Sin embargo, resulta caro adquirir sistemas que superen los 4 u 8 nodos y, por ello, los estudios de escalabilidad se quedan en el ámbito teórico.

Una solución para mejorar en escalabilidad, es usar salas de ordenadores y así conseguir un número de nodos superior a 30 ordenadores. Esta solución tiene un grave problema: la exclusividad. Si bien se podrían reservar durante varias horas, sin duda se deben liberar pasado ese tiempo para que otras asignaturas los puedan utilizar. Con esta restricción, los alumnos no pueden realizar sus ejercicios fuera de horas de clase, teniendo que reducir de manera importante la complejidad de los mismos. Por otro lado, la red de interconexión que utilizan las salas de ordenadores estaría compartida con otros sistemas del centro educativo y, sin duda, perjudicaría de manera importante la comunicación MPI de los ejercicios a ejecutar, haciendo muy complejo obtener tiempos de ejecución invariables de la carga de la red. Dada esta problemática, resulta poco fiable hacer estudios de *speedup* y escalabilidad a partir de ordenadores en red no dedicados y no exclusivos.

Nuestro objetivo es conseguir un entorno de programación MPI con dedicación exclusiva, con más de 250 procesadores y que resulte relativamente barato, entre 5.000 € y 6.000 €. Además, y aunque no forma parte de este estudio, nuestro recurso seguirá la actual tendencia de configurar y trabajar con sistemas con alta relación entre rendimiento y consumo. Así, se opta por sistemas con bajo consumo de energía.

2. Descripción del sistema

Siguiendo las líneas de especificación de la sección anterior, nuestro estudio se centra en sistemas de procesadores de bajo consumo y con una capacidad de cálculo razonable, que puedan ejecutar un sistema operativo (SO) completo desde el primer momento.

Si bien lo anterior incluye ordenadores personales, éstos se descartaron porque se quería un sistema que pudiese ser fácilmente aglutinado con otros, que ocupase el mínimo espacio posible y que tuviese un consumo de energía reducido. Del mismo modo, se descartaron sistemas empotrados/embebidos porque no permiten ejecutar fácilmente un SO Linux completo.

Así, se pasó a estudiar sistemas SoC (*System on Chip*) que tuviesen características parecidas a los basados en Raspberry Pi[1]. Este enfoque no es nuevo en las universidades españolas. Catalán *et al*. [1] en el Departamento de Informática e Ingeniería de Sistemas de la Universidad de Zaragoza, proponen un modelo de programación para Raspberry Pi basado en Arduino. De este modo, conectan un ordenador de sobremesa y dos placas Raspberry Pi a través de Internet para mostrar el funcionamiento de esas placas, aunque no desarrollan ninguna aplicación paralela. En cambio, Ortega *et al*. [3] del Departamento de Informática de la Universidad de Almería, seleccionan ARM como arquitectura de

[1] Raspberry Pi : `https://www.raspberrypi.org/`

Placa	Procesador	ARM	*Cores*	Frecuencia	MEM	Disco	Ethernet
Raspberry Pi-3B	Cortex-A53	v8	4	1.2 GHz	1 GB DDR2	MicroSD	Fast
Banana Pi-M2-Berry	Cortex-A7	v7	4	1 GHz	1 GB DDR3	MicroSD	Gigabit
Banana Pi-M3	Cortex-A7	v7	8	1.8 GHz	2 GB DDR3	eMMC	Gigabit
Orange Pi-plus-2E	Cortex-A7	v7	4	1.5 GHz	2 GB DDR3	eMMC	Gigabit
Odroid-XU4	Cortex-A15/A7	v8/v7	4/4	2/1.5 GHz	2 GB DDR3	eMMC	Gigabit
Odroid-C1+	Cortex-A5	v7	4	1.5 GHz	1 GB DDR3	eMMC	Gigabit
Odroid-C2	Cortex-A53	v8	4	1.5 GHz	2 GB DDR3	eMMC	Gigabit

Cuadro 1: comparativa SoC. Características técnicas relevantes

referencia para desarrollar los contenidos de la asignatura de Arquitectura de Computadores. Así pues, construyen un *cluster* de 4 Raspberry Pi para demostrar el potencial de la utilización de hardware de bajo coste para la resolución de problemas que requieren computación paralela.

Nosotros en este trabajo, queremos ir un paso más allá y construir un sistema de varias decenas de nodos que permita a los alumnos analizar la escalabilidad de los entornos de desarrollo MPI a un precio razonable.

2.1. Alternativas

A continuación, se describen las diferentes alternativas consideradas (cuadro 1) para seleccionar un sistema completo de procesador de bajo coste.

- Raspberry Pi-3B: es el diseño original y el más popular en sistemas SoC. Merece su estudio por tener un diseño claro y altas prestaciones. Se descarta por no disponer de Gigabit Ethernet y por su baja capacidad de memoria RAM.
- Banana Pi-M2-Berry: alternativa equivalente, que destaca por tener Gigabit Ethernet, pero tiene menos potencia de cálculo.
- Banana Pi-M3: opción superior a M2-Berry, con el doble de *cores* y memoria, almacenamiento eMMC más rápido, con un precio más elevado, pero no suficientemente potente.
- Orange Pi-plus-2E: con precio razonable, destaca por mejorar en memoria, almacenamiento y comunicación, pero poca potencia de cálculo.
- Odroid-XU4: es el sistema más potente, con 8 *cores* (4 Cortex A15 2.1 Ghz + 4 *cores* A7 1.5 Ghz) pero tiene un precio elevado.
- Odroid-C1+: sistema que compite directamente con Raspberry Pi-3B. Es una buena opción, pero el mismo fabricante ofrece un modelo superior más interesante y potente.
- Odroid-C2[2]: este sistema aglutina todas las ventajas de los sistemas anteriores a un precio razonable. Mejora en memoria, comunicación, almacenamiento y potencia de cálculo.

[2] `https://www.hardkernel.com/shop/odroid-c2`

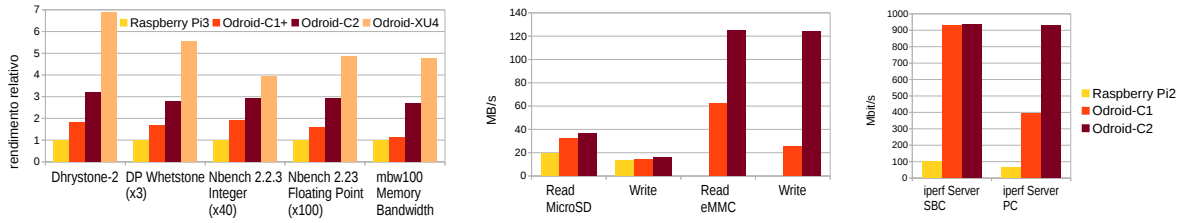
2.2. Placa seleccionada: Odroid-C2

La opción seleccionada ha sido: Odroid-C2. Para ver las prestaciones de la placa elegida, la figura 1 muestra una comparativa de ella con diversas alternativas: una placa más barata (Odroid-C1+), una placa más cara (Odroid-XU4) y una placa muy popular en el mercado (Raspberry Pi-3B). Estos valores se han obtenido de la página web del fabricante HardKernel® y muestran el rendimiento de la CPU (figura 1a), la velocidad de transferencia del acceso al almacenamiento (figura 1b) y la velocidad de transferencia de las comunicaciones (figura 1c).

Como se puede observar, las prestaciones de la placa escogida sobresalen del resto, a excepción de la placa XU4 que tiene un rendimiento superior, pero que tiene un coste económico de más del doble. La placa seleccionada tiene un precio de $46 con una muy buena relación rendimiento/precio. Sus características son:

- Amlogic ARM® Cortex®-A53(ARMv8) 1.5 GHz quad core CPUs
- Mali™-450 GPU (3 Pixel-processors + 2 Vertex shader processors)
- 2 GByte DDR3 SDRAM
- Gigabit Ethernet
- HDMI 2.0 4K/60Hz display
- H.265 4K/60FPS and H.264 4K/30FPS VPU
- 40 pin GPIOs + 7 pin I2S
- eMMC5.0 HS400 Flash Storage slot / UHS-1 SDR50 MicroSD Card slot
- 4 USB 2.0 Host, 1 USB OTG (power+data)
- Infrared(IR) Receiver
- Ubuntu 16.04 or Android 6.0 Marshmallow based on Kernel 3.14LTS

De entre todas sus características, destacamos la potencia de cálculo del ARMv8 (3 veces más potente que Raspberry Pi3), los 2 GB de RAM (2 veces superior), la velocidad en el acceso al almacenamiento en eMMC (120 MB/s) y la velocidad de transferencia en red de la Gigabit Ethernet (900 Mbit/s).



(a) rendimiento CPU (b) velocidad almacenamiento (c) velocidad Ethernet

Figura 1: comparativa placas SoC (*fuente del fabricante HardKernel*®)

2.3. Red de comunicación

La red de interconexión es uno de los factores más importantes que definen el rendimiento de los algoritmos paralelos de memoria distribuida. En este sistema no se ha optado por una solución de alto rendimiento, sino que se ha limitado a ofrecer una solución que obtenga el máximo rendimiento de las tarjetas de comunicación de las placas. Aunque la comunicación mediante USB puede ofrecer buenos rendimientos, estos están limitados en nuestro caso al estándar 2.0 cuya velocidad máxima es de 480 Mbps (siendo la velocidad efectiva de un dispositivo Gigabit-Ethernet USB 2.0 no superior a 300 Mbps). Dada esta limitación, se ha optado por comunicar las placas mediante la Gigabit-Ethernet incorporada, obteniendo así unas velocidades efectivas cercanas a los 900 Mbps.

Las diferentes placas se han interconectado mediante conmutadores dedicados. En nuestro caso, la forman dos conmutadores de la marca Zyxel modelo GS1900-48, que ha sido una solución suficiente y económica. Sus principales características son:

- Switching capacity (Gbps): 100
- Forwarding rate (Mpps): 74
- Packet buffer (byte): 1.5 M
- MAC address table: 8 K
- Jumbo frame (byte): 9 K

2.4. Montaje minimulticomputador

El minimulticomputador consta de: 64 Odroid-C2 con 8 GB de almacenamiento eMMC, 8 fuentes de alimentación con 10 salidas USB cada una de las fuentes, 2 conmutadores Zyxel y cables Ethernet categoría 5E. El montaje del minimulticomputador consistió en unir físicamente un Odroid al lado del otro y conectar cada uno de los Odroid a un puerto de un conmutador (figura 2). Como los conmutadores son de 48 puertos, se han usado 32 en cada uno de ellos para conectar los Odroid y un cable adicional para conectar los conmutadores entre si.

Por otra parte, se ha usado un Odroid extra para hacer de *frontend* del minimulticomputador. Este Odroid permitirá el lanzamiento de aplicaciones paralelas en los nodos y ofrecerá acceso de red a los mismos. Este nodo tiene los mismos componentes que los otros, pero añadiendo teclado, ratón, conexión HDMI a pantalla y Gibagit Ethernet por USB para disponer de acceso al minimulticomputador desde el exterior.

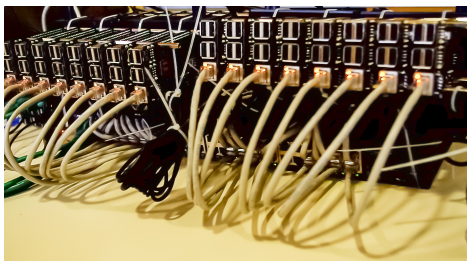


Figura 2: placas Odroid-C2 en funcionamiento

3. Entorno de desarrollo

Para que el minimulticomputador ejecute aplicaciones paralelas, se instala el SO Linux proporcionado por el fabricante (Ubuntu 16.4) grabable directamente en tarjetas de memoria eMMC o MicroSD. Esto nos permite utilizar un gestor de paquetes y, de esta forma, mantener actualizado el software e instalar los paquetes necesarios de manera muy sencilla.[3]

3.1. Software del nodo gestor

El sistema Linux instalado viene con el software estándar de Ubuntu para máquinas de sobremesa. Esto implica un gestor de arranque de servicios, un gestor de ventanas, etc. Se han añadido los siguientes servicios: servidor SSH (`openSSH`), servidor de nombres y servidor DHCP (`dnsmasq`), replicación de puertos, *tunneling* y *firewall* (`iptables`), servidor de ficheros en red (NFS).

Se instala también el paquete Open MPI para así poder ejecutar programas paralelos. Simplemente se

[3] Las imágenes se pueden obtener en `https://wiki.odroid.com/odroid-c2/os_images/ubuntu/ubuntu`

usan los comandos de Open MPI: `mpicc` y `mpirun`, el primero para compilar y el segundo para ejecutar. Este último tiene una opción que permite indicar las direcciones de los nodos que formarán parte de la ejecución paralela. Así, se crea un fichero de texto con los 64 nombres de intranet de los nodos.

3.2. Software de los nodos

En los nodos también se ha instalado el SO Linux Ubuntu 16.04 proporcionado por el fabricante, pero en este caso se han eliminado la mayoría de paquetes que no aportan nada en la ejecución paralela de aplicaciones, quedando así una instalación mínima que consume pocos recursos del sistema. Los servicios que se han mantenido han sido: servicio de acceso remoto (SSH), cliente de acceso a disco remoto (NFS) y el paquete de ejecución MPI (openMPI). Cualquier paquete que suponga una ocupación considerable de espacio en disco se ha desinstalado, como por ejemplo el escritorio Gnome.

Para poder administrar los nodos se ha permitido que los usuarios con privilegios (*root* y *sudoers*) puedan ejecutar comandos en los nodos desde el *frontend* sin necesidad de introducir cada vez la contraseña del usuario privilegiado. Esto se ha realizado mediante la generación de una clave RSA (`ssh-keygen`) y la posterior distribución de la clave pública a los nodos para que se pueda acceder sin solicitar la contraseña (`ssh-copy-id`).

Por otra parte, se ha añadido a los nodos, una entrada en el montaje del sistema de ficheros (`fstab`) para que, mediante el servicio NFS, puedan acceder al directorio de usuarios del nodo *frontend* (`/home`) y así compartirlo entre todos ellos. De este modo, en la ejecución paralela se ahorra el coste de transmisión de los códigos MPI binarios compilados. Esta transferencia se realiza por NFS y se aprovecha este protocolo para optimizar la distribución del código.

4. Verificación del sistema

Para verificar el correcto funcionamiento del sistema y que la refrigeración pasiva es suficiente, se ejecutaron pruebas de estrés con el paquete `stress`. Se comprobó que la temperatura de la CPU con los nodos inactivos estaba alrededor de 41 °C y que después de un periodo de estrés de las CPUs, las temperaturas se estabilizaban alrededor de 60 °C (algunos nodos hacia 56 °C y otros hacia 62 °C).

Dada la distribución física de las placas, se observa que si están apiladas horizontalmente, el calor de las inferiores perjudica a las superiores, llegando a temperaturas superiores a 75 °C. Aunque este valor no supone ningún problema en el sistema (el límite del fabricante está por encima de 80 °C) se observa que girando 90º la pila de Odroid y quedando en una distribución vertical (como en la figura 2) se reducen drásticamente las temperaturas, hacia 60 °C. Sin duda, se debe a que el calor generado se disipa por convección y no afecta tanto a las placas vecinas.

En la figura 3 se observa una placa exterior, donde la temperatura máxima en estrés se mantiene en 55,7 °C.

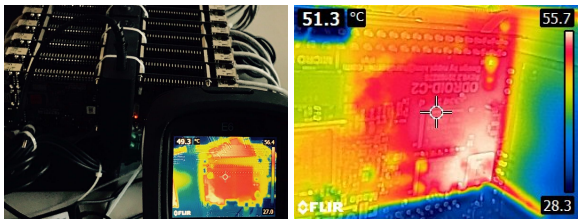


Figura 3: temperatura Odroid-C2 en pruebas de estrés

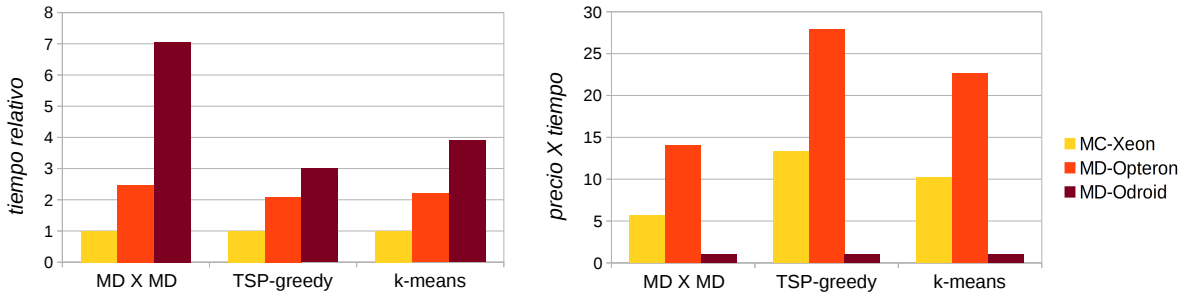
5. Ejemplos de ejecución MPI

En cada curso académico, y para consolidar los conceptos teóricos de la programación paralela, se seleccionan dos problemas diferentes que planteen de manera creciente cierta dificultad, siendo el primero más fácil que el segundo. Cada uno de estos ejercicios debe solucionarse con los estándares OpenMP (memoria compartida) y MPI (memoria distribuida).

Para este estudio mostramos tres posibles ejercicios: 1) producto de matrices dispersas, 2) búsqueda de un camino suficientemente óptimo del viajante de comercio y 3) cálculo de agrupamientos basado en el algoritmo *k-means*.

El primer problema calcula dos productos de matrices (figura 4, MD×MD): la primera, una matriz densa por una matriz dispersa y, la segunda, el producto de dos matrices dispersas. El ejercicio no presenta muchos problemas de paralelismo, ya que basta con repartir y replicar trozos de las matrices entre los procesos, para finalmente recoger los resultados y compactarlos en la matriz resultante. El tamaño del problema se ha dimensionado a matrices dispersas de 5.000×5.000 elementos.

El segundo problema es el "viajante de comercio" (figura 4, *TSP-greedy*) que debe visitar una serie de puntos unidos por caminos, de manera que los visite todos una sola vez, con el mínimo recorrido posible. En este caso se obtiene una solución suficientemente buena, pero no la solución óptima. Se basa en una selección *greedy* de todos los posibles inicios. La solución paralela pasa por repartir los posibles inicios y al final escoger el camino mínimo que haya calculado uno de los procesos. El número de sitios a visitar para este ejemplo es de 3.000.



(a) tiempo relativo

(b) coste: precio × tiempo

Figura 4: ejecución secuencial

Por último, el algoritmo *k-means* (figura 4, *k-means*) trata de crear agrupaciones de elementos generados aleatoriamente. Busca de forma iterativa k-centroides de los elementos a tratar, hasta que cada uno de ellos se estabilice y quede asignado a un centroide. Los centroides son calculados a partir de la media de los elementos que le pertenecen. En este caso hay una recurrencia en cada iteración, que obliga a sincronizar los procesos para actualizar correctamente los centroides. En este ejercicio se han buscado 200 centroides a partir de 400.000 elementos.

En la figura 4a se muestra el tiempo de ejecución secuencial relativo del mismo algoritmo usando únicamente un procesador de cada sistema. La máquina con procesadores Xeon es la más rápida, siendo hasta 2,5 veces más rápida que la máquina con procesadores Opteron y hasta 7 veces más rápida que un procesador ARM de una placa Odroid-C2.

Sin embargo, cuando tenemos en cuenta el precio de un sistema con un procesador de cada tipo, el coste relativo de la ejecución de estos ejercicios cambia significativamente, siendo un Odroid-C2 unas 10 veces mejor que un sistema con procesador Xeon, y unas 20 veces mejor que un sistema con procesador Opteron. La figura 4b muestra el coste, calculado como el tiempo de ejecución multiplicado por el precio[4] unitario con un sistema monoprocesador, es equivalente al coste calculado como precio dividido por el rendimiento (tal como se verá en las figuras de la siguiente sección).

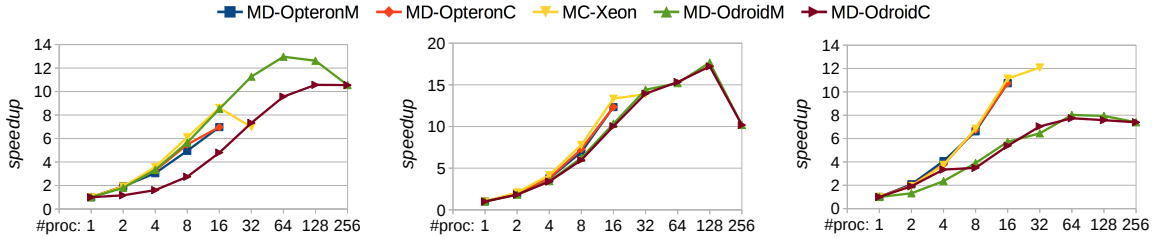
6. Análisis de rendimiento

Los códigos anteriores se han paralelizado con el estándar MPI y se han ejecutado mediante un sistema de colas (para garantizar exclusividad) en las diferentes máquinas disponibles en la asignatura. Mostramos únicamente aquellas que nos sirven para evaluar las diferencias con MPI. Una es un servidor de memoria compartida con dos procesadores Intel Xeon E5-2660 a 2.2 GHz y 32 GB de RAM (MC-Xeon). Otra son 4 servidores formando un *cluster* de memoria distribuida, con 2 procesadores Opteron 2210 y 4 GB de RAM cada uno (MD-Opteron). Y, por último, el sistema objeto de este trabajo, que lo forman 64 placas Odroid-C2 con procesador ARM Cortex A-53 a 1.5 GHz y 2 GB de RAM cada una (MD-Odroid).

La figura 5 muestra la evolución del *speedup* a medida que se van utilizando más procesadores. Para MD-Opteron y MD-Odroid se presentan dos ejecuciones con mapeos de procesos diferentes. MD-OpteronM y MD-OdroidM equivale a un mapeo pensando en problemas que sean *Memory-Bound*, donde se producen muchos accesos a memoria por parte del algoritmo y se intenta que un proceso tenga la memoria para él solo. Así, para MD-OpteronM una ejecución de 4 procesos asignará un proceso en cada nodo, ocupando de esta forma los 4 nodos. En MD-OdroidM cada placa ejecuta un único proceso, desde la ejecución de 2 procesos hasta 64 procesos (para 128 y 256 ya se asignan 2 y 4 procesos por placa). MD-OpteronC y MD-OdroidC equivale a un mapeo pensado en problemas *CPU-Bound*, donde no se hacen demasiados accesos a memoria y el cuello de botella se debe al uso de la CPU. En este caso se intenta asignar el máximo de procesos a cada nodo, de forma que la posible comunicación entre los procesos de un nodo se haga mediante memoria compartida, sin necesidad de utilizar la red de interconexión. Por ejemplo, para 4 procesos sólo se utilizará un nodo y, a medida que se incrementa el número de procesos se van utilizando más nodos.

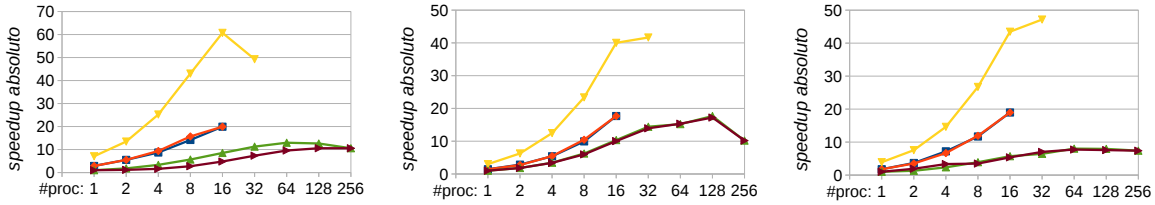
En MD×MD (figura 5a) las máquinas tienen una evolución similar, pero sin obtener *speedup* muy eficientes. La reducción de la eficiencia se debe sobretodo a que es un algoritmo *Memory-Bound* y a la necesidad de comunicar los resultados de la multiplicación al nodo recolector. Destaca la pérdida

[4]Los precios varían mucho según el modelo concreto de procesador, pero se considera un precio de 2.000 € para un sistema con un procesador Xeon y otros 2.000 € para un sistema con procesador Opteron. Por contra, el precio de una placa Odroid-C2 es de unos 50 €.



(a) MD × MD (b) TSP-*greedy* (c) *k-means*

Figura 5: evolución del *speedup* y escalabilidad



(a) MD × MD (b) TSP-*greedy* (c) *k-means*

Figura 6: *speedup* absoluto

de rendimiento en 128 y 256 del sistema MD-OdroidM y el poco rendimiento que se obtiene en la versión MD-OdroidC, debido a la competición por la memoria que se produce en cada placa por el hecho de tener 4 procesos asignados.

Con el algoritmo TSP-*greedy* (figura 5b) vemos una evolución similar pero con *speedup* más eficientes. Este algoritmo es del tipo *CPU-Bound* y tiene un mayor grado de paralelismo. Sólo es necesaria la comunicación al final de la ejecución, para decidir qué proceso ha encontrado el mejor camino y entonces transmitirlo al nodo recolector. Los dos tipos de mapeo dan resultados equivalentes.

Por último, observamos el algoritmo *k-means* (figura 5c) que se comporta de una forma similar en MC-Xeon y MD-Opteron(MC), pero tiene una pérdida de rendimiento importante en los sistemas MD-Odroid(MC). En este caso se necesita una sincronización en cada iteración para recolectar y distribuir los nuevos centroides. Aquí se ve como el hecho de tener multiprocesadores en MC-Xeon y MD-OpteronC, donde algunos procesos comparten memoria, la comunicación entre ellos no utiliza la red. Sin embargo, los MD-Odroid(MC) tienen que usar más la red de interconexión para comunicar los centroides, resultando en una pérdida de rendimiento importante. Destaca el beneficio inicial (hasta 4 procesos) de MD-OdroidC sobre MD-OdroidM ya que la comunicación en este caso también es interna al nodo, sin necesidad de utilizar la red de interconexión.

En cualquier caso, se puede observar que, al trabajar con más nodos, salen a relucir de forma más evidente los problemas y retardos producidos por la necesaria comunicación entre los procesos de un algoritmo paralelo. Este comportamiento es beneficioso, desde el punto de vista educativo, porque permite una experiencia más realista en los alumnos.

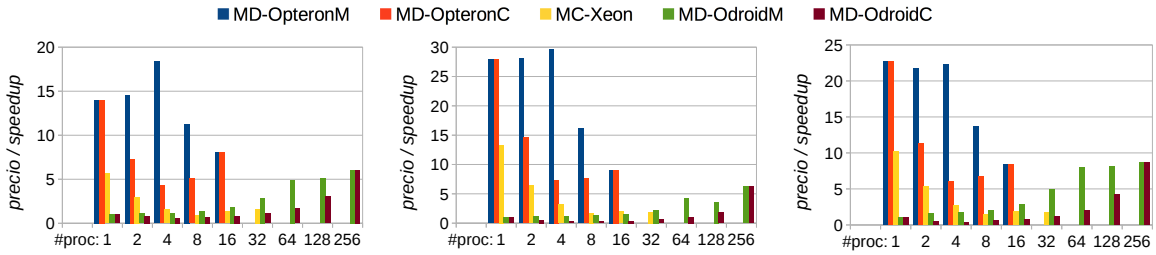
En la figura 6 se puede observar el *speedup* absoluto de las tres máquinas (tomando como base la más lenta). Se hace evidente que la máquina de memoria compartida MC-Xeon es la que tiene mejor rendimiento, dada la potencia de cálculo de cada *core* del sistema. Después, el sistema de memoria distribuida MC-Opteron ofrece mejor rendimiento que MC-Odroid. Este último es el peor en rendimiento de los tres.

Sin embargo, cuando añadimos el precio a la función de coste (figura 7), calculando el coste en dinero relativo al rendimiento absoluto que ofrece cada sistema, los resultados cambian drásticamente.

Como se ha visto, el sistema MC-Xeon es el que tiene mejor rendimiento, debido a su potencia de cálculo y comunicación interna, presentando así una relación precio/rendimiento muy buena, sobretodo para ejecuciones de 8 a 16 procesos.

Los sistemas MD-Opteron, en cualquier configuración y mapeo, suponen el mayor gasto por rendimiento de los tres. No hay ninguna combinación que indique, respecto a los demás, que sea una opción óptima cuando se tiene en cuenta la relación precio/rendimiento. Para mapeos *Memory-Bound* (MC-OpteronM) la situación es aún peor, porque el precio del sistema para ejecutar 4 procesos implica el coste de los 4 nodos.

Por último destacar que los sistemas MC-Odroid(MC) son los que ofrecen la mejor relación precio/rendimiento. El motivo es sobretodo por su bajo coste. El hecho de ir incrementando el



(a) MD × MD (b) TSP-*greedy* (c) *k-means*

Figura 7: precio / *speedup* absoluto

coste, añadiendo placas cada vez que se quieren ejecutar más procesos, hace que en cualquier configuración y mapeo ganen al resto por una amplia diferencia. Sólo para mapeos de 4 procesos por nodo y ejecuciones de 64, 128 y 256 procesos, tienen una relación precio/rendimiento equivalente a los otros dos, pero en el resto son la mejor opción. De los diferentes tipos de mapeo, MC-OdroidC da peores rendimientos para algoritmos *Memory-Bound* (figura 7a) que la opción MC-OdroidM. Pero, a pesar de ello, MD-OdroidC sigue obteniendo mejor relación precio/rendimiento que la otra, ya que aprovecha al máximo los *cores* de cada placa.

Si añadimos a la fórmula el consumo de energía, también acaban ganando los sistemas Odroid. Cada placa tiene un consumo en ejecución de unos 5 W (2 W en *idle*) que en total supone un máximo de 320 W. El sistema MC-Xeon consume de 110 W (*idle*) a 350 W como máximo, y cada nodo del sistema MD-Opteron consume 350 W, que supone un máximo de 1.400 W. También por el hecho de ser incremental, la opción de los sistemas MD-Odroid ofrece la mejor relación consumo/rendimiento.

7. Conclusiones

Este trabajo propone un minimulticomputador de bajo coste para usarlo en prácticas de programación paralela. El sistema basado en Odroid-C2 tiene un coste equivalente a un servidor pequeño con memoria compartida y permite el acceso a una mayor cantidad de procesadores. Por otra parte, tratar de igualar el número de procesadores en sistemas con servidores de memoria compartida unidos por red de interconexión, tendría un coste mucho más elevado. En cualquier caso, el rendimiento que se obtiene en relación al precio y al consumo es excelente.

Cabe destacar que este sistema se puede usar en otros ámbitos educativos, donde se precisen de múltiples nodos para realizar sus prácticas. Por ejemplo, en asignaturas de Sistemas Distribuidos (*cloud computing*, etc.), asignaturas de Administración de Sistemas (`kubernetes`, etc.) y asignaturas de Administración de Redes (conmutadores, etc.). Del mismo modo, creemos que también puede ser interesante aplicarlo en asignaturas optativas de estudios universitarios, al igual que en asignaturas de módulos superiores de secundaria. Todo ello haciendo que la inversión necesaria no suponga una debacle para el presupuesto docente.

El valor añadido de este entorno supone acercar a los alumnos a un sistema con más procesadores y más opciones de intercomunicación, para así poder hacer estudios de escalabilidad más realistas. Todo ello manteniendo un buen compromiso entre prestaciones, precio y consumo.

Agradecimientos

Este trabajo ha contado con la financiación del Gobierno de España bajo los contratos TIN2016-77836-C2-1-R, TIN2016-77836-C2-2-R, TIN2016-75344-R y DPI2016-77415-R, y también de la Generalitat de Catalunya como Grupos de Investigación Consolidados 2017-SGR-688 y 2017-SGR-990.

Referencias

[1] Carlos Catalán Cantero, Alfonso Blesa Gascón. Enseñanza de sistemas empotrados: de Arduino a Raspberry Pi. En *Actas de las XXII Jenui*, páginas 351-354, Almería, julio 2016.

[2] Domingo Giménez. Un Curso práctico de programación paralela basado en problemas de Concurso Español de Programación Paralela. En *Actas de las XXII Jenui*, páginas 19-26, Almería, julio 2016.

[3] G. Ortega, J.M.G. Salmerón, C. Medina-López, J.L. Redondo, J.F.R. Herrera, N.C. Cruz, G. Barrionuevo, P.M. Ortigosa, V. González-Ruiz, E.M. Garzón. Procesadores de bajo coste y su aplicación en la docencia de Ingeniería de Computadores. En *Actas de las XXII Jenui*, páginas 343-349, Almería, julio 2016.

[4] J. Santamaría, M. Espinilla, A.J. Rivera, S. Romero. Potenciando el aprendizaje proactivo con ILIAS&WebQuest: aprendiendo a paralelizar algoritmos con GPUs. En *Actas de las XVI Jenui*, páginas 503-506, Santiago Compostela, julio 2010.