Facultad de Informática

Adrián García García

Instrumentación del runtime de OpenMP

24 de junio de 2017

Índice

1. Introducción

Este trabajo tiene como objetivo realizar una instrumentación del runtime de OpenMP para permitir la recolección de métricas de rendimiento en los programas que hacen uso de esta API de paralelización. La conferencia relacionada con este trabajo estaba orientada más hacia el paradigma de computación basado en tareas y la paralización de diferentes programas que se ejecutan en procesadores simétricos convencionales. Sin embargo, en el ámbito de investigación en el que participo junto al profesor Juan Carlos Sáez Alcaide hemos observado que la gran mayoría de benchmarks multihilo de suites comerciales (SPEC, PARSEC, Minebench, etc.) hacen uso de directivas de paralelización convencionales como *#pragma parallel for*. Con el objetivo de evaluar el rendimiento de estos benchmarks, he realizado una instrumentación del runtime de OpenMP y más adelante presentaré un caso de estudio para exponer diferentes retos que debe afrontar su runtime en sistemas multicore asimétricos (AMP).

La parelización basada en directivas de compilación que reparten el trabajo de los bucles realiza una división del espacio de iteraciones en fragmentos de igual tamaño para cada hilo. No obstante, esta aproximación trae significativos problemas de rendimiento y justicia en los sistemas AMP, ya que sus cores funcionan a diferente frecuencia y pueden presentar características microarquitectónicas diversas (diferentes tamaños de cache, ejecución en orden o fuera de orden, etc.). Estas diferencias provocan que un hilo que se ejecuta en un core big generalmente termine antes su asignación de trabajo. Cuando sucede esto, estos hilos se suelen bloquear en una barrera de memoria a través de una espera activa con el objetivo de sincronizarse con el resto de hilos que se ejecutaban en cores small. Considero que esta situación representa un desperdicio de recursos debido a dos motivos principales:

- Los sistemas operativos de propósito general no son capaces de identificar a los hilos que realizan una espera activa y, por tanto, su planificador no puede intercambiar estos hilos por otros que sean capaces de hacer un uso efectivo de los cores.

- En el contexto de monitorización del rendimiento y recogida de métricas para la toma de decisiones de planificación, las técnicas de espera activa como mecanismo de sincronización generan unos valores de rendimiento engañosos. Esto se debe a que los hilos que usan estas técnicas producen

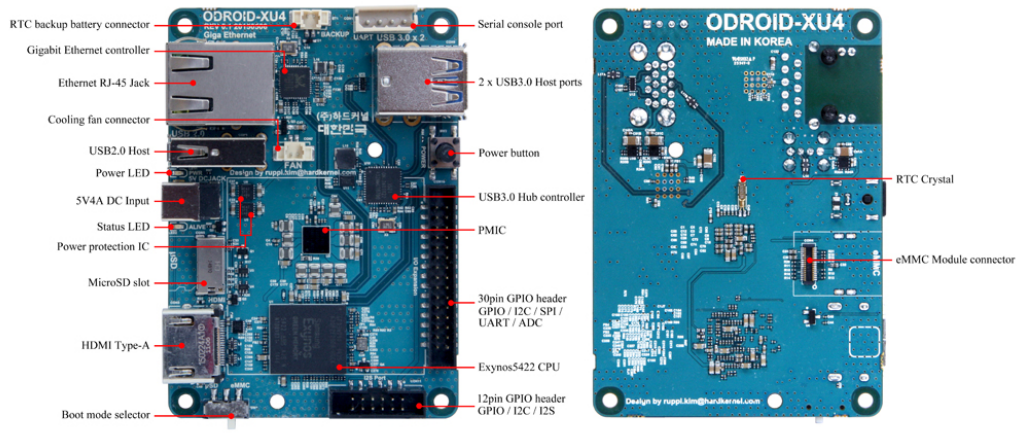

Figura 2: Placa ODROID-XU4

unos valores altos de IPC, cuando en realidad se encuentran ejecutando instrucciones inútiles para el progreso de la aplicación. Estas medidas pueden llevar a tomar decisiones erróneas al planificador o incluso a realizar cargos adicionales en plataformas de Cloud Computing en las que se factura a los usuarios en base al tiempo de uso de CPU y a diferentes métricas recogidas en tiempo de ejecución.

Con el objetivo de recoger información de diferentes métricas de rendimiento he elegido utilizar PMCTrack [1], se trata de una herramienta de monitorización del rendimiento a través de contadores hardware para Linux. Esta herramienta ha sido diseñada especialmente para ayudar a los desarrolladores del kernel en la implementación de algoritmos de planificación que utilizen los datos de los contadores de monitorización del rendimiento (Performance Monitoring Counters - PMCs) para realizar optimizaciones en tiempo de ejecución. A pesar de ser una herramienta orientada al sistema operativo, PMCTrack también está equipada con una serie de componentes en espacio de usuario (bibliotecas, entorno gráfico y de línea de comandos) que permiten obtener información de monitorización de aplicaciones en ejecución. En este trabajo se utilizará la biblioteca de PMCTrack para instrumentar el código del runtime de OpenMP y poder recoger diferentes métricas de los bucles paralelizados con este framework. Se puede encontrar un manual de usuario junto a una guía de instalación en la página web oficial del proyecto [2].

Para la realización de este trabajo se ha empleado la placa de desarrollo

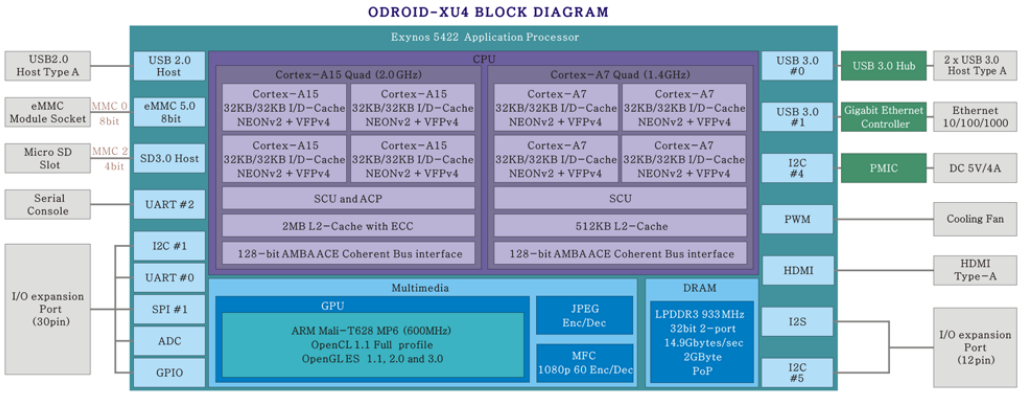

Figura 3: Diagrama de bloques de la placa Odroid XU4

Odroid XU4 [3], el principal motivo de su elección es que incorpora un procesador multicore asimétrico y, además, ya contaba con experiencia trabajando con ella en el Trabajo de Fin de Grado y anteriores investigaciones. Esta placa está provista de un SoC(*System-On-Chip*) Exynos 5422 de Samsung fabricado en 28nm, que integra un procesador ARM big.LITTLE de ocho núcleos. Estos se dividen en dos clusters, cuatro cores Cortex A15 (2GHz) con ejecución fuera de orden y 2MB de último nivel de cache (L2) y otros cuatro cores Cortex A7 (1.4GHz) con ejecución en orden y 512KB de LLC (L2). Ambos comparten espacio de direcciones para acceder a una memoria principal DDR3 de 2GB que funciona a 750 MHz. Para mostrar la configuración de la placa con mayor detalle, la figura 3 muestra el diagrama de bloques de la placa Odroid XU4.

El resto del trabajo está estructurado de la siguiente forma, en la Sección 2 se describe de forma detallada el entorno experimental y los pasos seguidos para su construcción. En primer lugar, se explica cómo usar un compilador cruzado y por qué es necesario en la Sección 2.1. A continuación, en la Sección 2.2 se describe el proceso seguido para instrumentar el runtime de OpenMP y en qué partes del código fue necesario introducir llamadas a la librería de PMCTrack para recoger métricas de la paralelización de bucles en tiempo de ejecución. Más adelante, en la Sección 2.3 se expone la forma de activar la instrumentación para ejecutar un benchmark y medir el rendimiento de su ejecución con diferentes métricas. Por último, en la Sección 3 se presentan los resultados obtenidos al instrumentar la ejecución de benchmarks paralelos en la placa Odroid XU4 y en la Sección 4 se exponen las

conclusiones que he obtenido de este trabajo.

2. Entorno experimental

2.1. Construcción de un compilador cruzado

2.2. Instrumentación del runtime de OpenMP

En esta sección se explicará el procedimiento seguido para instrumentar el runtime de OpenMP usando la librería de PMCTrack. Con el fin de aprender a usar esta librería, se consultaron los ejemplos disponibles de forma pública junto a las fuentes del proyecto en el repositorio oficial de PMCTrack en github [4].

2.3. Recolección de métricas

3. Resultados

4. Conclusiones

Referencias

[1] *An OS-oriented performance monitoring tool for multicore systems*. Saez, J. C., Casas, J., Serrano, A., Rodríguez-Rodríguez, R., Castro, F., Chaver, D., & Prieto-Matias, M. (2015, August). In European Conference on Parallel Processing (pp. 697-709). Springer International Publishing.

[2] *Manual de usuario y guía de instalación*. Página web oficial.

[3] *Odroid XU4: User Manual*. Hardkernel. Página web.

[4] *Repositorio oficial de PMCTrack*. Ejemplos de código utilizados disponibles en *pmctrack/test/test_libpmctrack/*. Enlace a git.