Infraestructura de Análisis de Rendimiento

Propuesta de tesis para

Magister en Cómputo de Altas Prestaciones

Universidad Nacional de La Plata
Facultad de Informática

Tesista: Andrés More - amore@hal.famaf.unc.edu.ar
Director: Dr. Fernando G. Tinetti - fernando@lidi.info.unlp.edu.ar

Abril de 2015

1. Objetivo

La propuesta principal consiste en el desarrollo de una infraestructura de soporte para el análisis de rendimiento en aplicaciones de cómputo de altas prestaciones.

El problema a resolver es simplificar la compleja tarea de realizar un análisis de rendimiento sobre programas de simulación y cálculo científico. El análisis implica la ejecución repetitiva de varios casos de prueba bajo diferentes condiciones, además de la aplicación de múltiples herramientas para analizar el comportamiento y la utilización de los recursos disponibles.

La infraestructura a desarrollar implementará un procedimiento sistemático de análisis de rendimiento ejecutando pruebas de referencia, herramientas de perfil de rendimiento y análisis de resultados. La infraestructura generará como resultado final un informe detallado para soportar la tarea de optimización con información cuantitativa. El reporte final incluirá datos estadísticos de la aplicación y el sistema donde se ejecuta, además de gráficos de desviación de resultados, escalamiento de problema y capacidad de cómputo e identificación de cuellos de botella.

2. Motivación/Estado del Arte del Tema

En el área de HPC los programadores son los mismos especialistas del dominio del problema a resolver. Las rutinas más demandantes de cálculo son en su mayoría científicas y su alta complejidad hace posible su correcta implementación sólo por los mismos investigadores. Estas cuestiones resultan en un tiempo reducido de optimización de rendimiento e impactan directamente en la productividad de los grupos de investigación y desarrollo. Frecuentemente el proceso de optimización termina siendo hecho de modo *ad-hoc*, sin la utilización de información cuantitativa para dirigir los esfuerzos de mejora.

Algunas referencias a la problemática se incluyen a continuación como soporte de la propuesta:

1. Cómo correctamente mostrar resultados de rendimiento por *Fleming, Philip J. and Wallace, John J.* en *How Not to Lie with Statistics: the Correct Way to Summarize Benchmark Results*, ACM, 29/3, 1986. Un resumen apropiado de los resultados de ejecución de una aplicación facilita la tarea de entender su comportamiento bajo diferentes circunstancias.

2. Cómo acceder a memoria eficientemente por *Drepper, Ulrich* en *What Every Programmer Should Know About Memory*, Reporte Técnico, 2007. Conocer en detalle el funcionamiento de la memoria de un sistema, incluyendo sus distintos niveles de jerarquía y su funcionamiento a bajo nivel permite aprovechar al máximo su capacidad de transferencia y realizar cómputo eficientemente.

3. Una discusión de la dificultad de la programación en paralelo por *McKenney, Paul E.* en *Is Parallel Programming Hard, And, If So, What Can You Do About It?*, Reporte Técnico, 2007. Las posibilidades de mejora son limitadas si solo se busca mejorar un algoritmo o utilizar una máquina más rápida, dejando como única opción la paralelización para distribuir el cómputo entre diferentes sistemas.

4. Una revisión de como medir rendimiento durante el desarrollo de una aplicación por *Woodside, Murray and Franks, Greg and Petriu, Dorina C.* en *The Future of Software Performance Engineering*, IEEE Computer Society, 2007. La medición de rendimiento es una tarea no trivial, que realizada sin la correcta disciplina puede llevar a conclusiones erróneas.

5. Ejemplos de optimización compleja por *Jeff A. Bilmes et al.* en *BLAS3 Compatible Fast Matrix Matrix Multiply* y también por *R. Clint Whaley et al.* en *Automatically Tuned Linear Algebra Software*, ICSI, 1998. La tarea de optimización requiere tener en cuenta detalles complejos del funcionamiento de la arquitectura de bajo nivel de los sistemas de cómputo.

6. Cómo utilizar información de contadores de *hardware* durante la ejecución de un programa por *Thomas Ball* en *Eficiently counting program events with support for on-line queries*, ACM Programming Languages and Systems, 1994. Los procesadores modernos facilitan entender su rendimiento mediante contadores programables, permitiendo conocer el uso de sus componentes internos.

7. Cómo optimizar software teniendo en cuenta arquitectura de bajo nivel por *Intel* en *Intel® 64 and IA-32 Architectures Optimization Reference Manual*, Intel Press, 2013. Las arquitecturas Intel y compatibles son las más usadas actualmente, las reglas básicas recomendadas para un funcionamiento eficiente por el mismo fabricante permiten mejorar el rendimiento.

Además de la teoría necesaria para entender el rendimiento de un programa, se necesita además tener experiencia en la aplicación de diversas herramientas que permiten extraer información cuantitativa sobre el rendimiento de una aplicación, este trabajo de investigación revisará las disponibles actualmente y se centrará en la automatización de su aplicación general para focalizar los esfuerzos de una optimización posterior, maximizando su impacto.

En principio el desarrollo se enfoca en ambientes *GNU/Linux*, con aplicaciones utilizando paralelismo implementado con tecnología *OpenMP*.

3. Temas de Investigación

Esta sección contiene una lista inicial de los temas a trabajar:

Análisis de Rendimiento Sistemático: Realizar un análisis de rendimiento es algo no trivial y el no poseer un procedimiento sistemático implica perdida de tiempo en la identificación de las tareas a realizar y que información obtener de las mismas.

Configuración y ejecución de pruebas de referencia: Las pruebas de rendimiento proveen una línea base de comparación sobre las capacidades de los diferentes componentes de un sistema de cómputo, ya sea aislados o trabajando en conjunto. Las principales candidatas son *HPC Challenge Benchmark (HPCC)* y *High Performance Linpack (HPL)*. HPCC incluye un conjunto de pruebas que sintetizan diferentes casos de uso del mundo real. HPL mide el poder de cómputo en la cantidad de operaciones de número flotante que un sistema realiza al resolver un sistema linear de ecuaciones. Aunque HPCC es la que brinda información más completa y variada, HPL es utilizada mundialmente para comparar el rendimiento de diferentes sistemas.

Aplicación general de herramientas de perfil de ejecución: Un perfil de ejecución de una aplicación permite conocer su comportamiento interno, identificando que partes necesitan ser optimizadas. El perfil puede ser construido utilizando instrumentación del código e incluso utilizando soporte de contadores de *hardware*. Los contadores a nivel de *hardware* ofrecen mayor precisión, evitan errores de muestreos y producen menor sobrecarga que instrumentar *software*. Las herramientas candidatas a utilizar para obtener esta información son *GNU gprof* y *perf*. GNU gprof es una herramienta de instrumentación y muestreo que introduce código durante la compilación para obtener información de rendimiento. *Performance Counters for Linux (perf)* es una herramienta de análisis estadístico de rendimiento a nivel de sistema, en particular utilizando contadores de *hardware* a nivel de CPU.

Generación de gráficos de rendimiento: Los gráficos facilitan el análisis y modelado del comportamiento de una aplicación. Por ejemplo: un histograma de la distribución de resultados permite entender la desviación de los mismos, un gráfico histórico comparando diferentes versiones de una aplicación permite comprobar mejoras, un gráfico de tiempos de ejecución permite entender escalamiento de una aplicación. Para la generación de gráficos se planea utilizar la librería *matplotlib*.

Generación de reportes: Como soporte para el análisis de rendimiento, se incluye como resultado final un reporte integral con información de referencia, datos de perfil de ejecución y gráficos de rendimiento. Identificando problemas y áreas de mejora potenciales.

4. Desarrollos/Trabajo Experimental a Realizar

La infraestructura a desarrollar consiste de distintos componentes, como se propone en la Figura 1 a continuación.

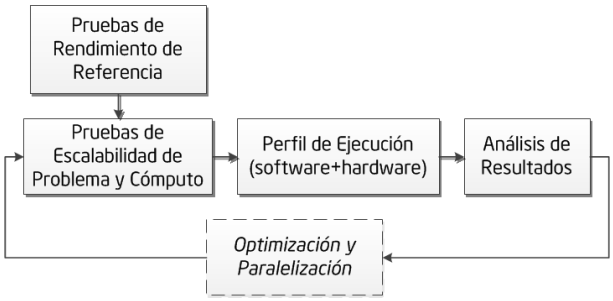

Figura 1: Infraestructura a Desarrollar

Pruebas de Rendimiento de Referencia: este componente automáticamente configura y ejecuta pruebas sobre el sistema, publicando números de rendimiento relevantes.

Pruebas de Escalabilidad: este componente ejecuta la aplicación variando tamaño de problema y también capacidad de cómputo para obtener información de comportamiento y límites de escalabilidad.

Perfil de Ejecución: este componente automáticamente configura y ejecuta herramientas para medir rendimiento e identificar cuellos de botella en una aplicación utilizando soporte de *software* y *hardware*.

Análisis de Resultados: utilizando la información anterior, este componente calcula límites de potenciales mejoras y lista los cuellos de botella principales asociados al código de la aplicación.

Optimización: a partir de la información anterior, este último componente compila un reporte incluyendo gráficos y referencias al código específico a optimizar.

La tarea de optimización y paralelización entonces es realizada manualmente, utilizando información estadística y cuantitativa sobre el comportamiento de la aplicación, extraída automáticamente por la infraestructura a desarrollar.

Las secciones ya identificadas a incluir en el reporte de salida son las siguientes:

Información de Hardware: modelo de CPU, cantidad de unidades de procesamiento, composición de la memoria del sistema (Cache y RAM).

Información del Programa: nombre, ubicación, parámetros de entrada, pasos de compilación.

Informacion de Software: núcleo del sistema, librerías principales

Sanidad del Programa: resultado de una ejecución rápida

Composición del Programa: tamaño del programa, análisis de las estructuras que lo componen

Pruebas de Rendimiento: resultados de la ejecución de HPCC y HPL

Caso de Prueba: histograma con los tiempos de ejecución de múltiples ejecuciones.

Escalamiento del Problema: gráficos de escalamiento de unidades de cómputo y de tamaño del problema.

Optimizaciones: grafico comparando el comportamiento del programa bajo diferentes niveles de optimización en el compilador.

Perfil de Rendimiento: perfil del programa incluyendo el tiempo de ejecución mapeado al código fuente.

Utilización de Recursos: perfil del uso de los recursos del sistema como memoria, CPU y entrada/salida.

Código Anotado: código fuente asociado a instrucciones de *hardware* utilizadas y su impacto en la ejecución global.

Vectorizaciones: reporte de vectorización de ciclos de cómputo.

Contadores de Hardware: reporte genérico del estado de lo contadores de *hardware*.

Una vez disponible toda la información detallada anteriormente, se puede proceder diferentes técnicas de optimización para mejorar el rendimiento de un programa. Los métodos usuales de optimización a bajo nivel son los siguientes:

Código: se analiza el código fuente para mejorar la predicción de saltos, realizar inlining de rutinas muy utilizadas, alinear código y realizar unrolling de ciclos cuando permite mejorar el rendimiento.

Ejecución: se analiza el código a nivel de ensamblador para comprobar el uso de instrucciones livianas o vectoriales, y además un uso reducido de registros.

Memoria: se analiza las estructuras de datos para incrementar la tasa de transferencia a memoria, minimizar fallas de cache, alinear datos y mejorar la localidad del acceso a los mismos.

Precarga: se analiza el uso de instrucciones de *hardware* para precargar datos en cache cuando los algoritmos predictivos no son suficientes.

Punto Flotante: se considera el relajamiento de las reglas de redondeo y representación según los estandares, intercambiando un mayor error acumulado por mejor velocidad de procesamiento.

5. Esquema de Plan de Trabajo

El siguiente cronograma muestra el plan de actividades tentativo incluyendo actividades y tiempos.

Cuadro 1: Detalle de Actividades

Actividad	**Duración**
Estado del Arte	Abril
Diseño de Alto Nivel	Mayo
Prototipo	Junio
Pruebas de Rendimiento	Julio
Cálculo de Leyes	Agosto
Perfil de Rendimiento	Septiembre
Eventos y Vectorización	Octubre
Documentación Tesis	Noviembre

6. Posibilidades de Realización

El alumno tiene a su disposición acceso al equipamiento necesario como parte de su ámbito laboral. El alumno trabaja como Ingeniero en Software en *Argentina Software Design Center* (ASDC - Intel Córdoba); también como docente e investigador en el Instituto Universitario Aeronáutico dictando tanto cursos de grado (Cómputo de Altas Prestaciones) como postgrado (Implementación de Sistemas Operativos).

7. Bibliografía Básica Relacionada

Un conjunto inicial a modo de referencia es incluido en la bibliografía listada a continuación.

Referencias

[1] J. Browne, *A critical overview of computer performance evaluation*, 1976.

[2] G. Mattson, B.A. Sanders and B.L. Massingill, *Patterns for Parallel Programming, Addison-Wesley*, 2004.

[3] T. Margalef, J. Jorba, O. Morajko, A. Morajko, E. Luque, *Different approaches to automatic performance analysis of distributed applications*, 2004.

[4] C. Smith, *Introduction to software performance engineering: origins and outstanding problems*, 2007.

[5] M. Woodside, G. Franks, D. Petriu, *The Future of Software Performance Engineering*, 2007.

[6] K. Huck, O. Hernandez, V. Bui, S. Chandrasekaran, B. Chapman, A. Malony, L McInnes, B. Norris, *Capturing performance knowledge for automated analysis*, 2008.

[7] Andres More, *Herramientas de Soporte para Análisis de Rendimiento*, Trabajo Final Especializacion HPC/GRID - UNLP 2013.

[8] Fernando G. Tinetti, Mariano Mendez, and Armando De Giusti, *An Automated Approach to Hardware Performance Monitoring Counters*, 2013.

[9] OpenMP Architecture Review Board, *OpenMP Application Program Interface Version 3.0*, Mayo 2008.