

Arquitectura de Sistemas

Análisis de rendimiento

Gustavo Romero López

Updated: 15 de septiembre de 2022

Arquitectura y Tecnología de Computadores

- ⊙ Aprender los principios de la optimización de código:
 1. Escoger el mejor algoritmo.
 2. Utilizar la mejor implementación.
 3. Optimizar.
- ⊙ Instrumentación: código analizándose sí mismo.
 - Propensa a errores.
 - ¿Mide lo que en realidad creemos?
 - Intentad evitarla.
- ⊙ Software de análisis de rendimiento:
 - Instrumentación: gprof
 - Simuladores: valgrind
 - Muestreo (estadísticos): oprofile y perf

- ⊙ Híbrido entre instrumentación y muestreo.
- ⊙ Forma de uso:
 1. Compilar con la opción “-pg”.
 2. Al ejecutar se crea un fichero llamado `gmon.out`.
 3. Analizar los resultados de `gmon.out` con `gprof`.
- ⊙ Tutorial:
<http://www.thegeekstuff.com/2012/08/gprof-tutorial/>

- ⊙ Máquina virtual que emplea técnicas de interpretación dinámica (JIT).
- ⊙ Colección de herramientas para detección de errores y análisis de rendimiento.
- ⊙ Inconvenientes:
 - Ejecución entre 5 y 20 veces más lenta.
 - Alta utilización de memoria.
- ⊙ Ventajas:
 - No es necesario modificar los binarios.
 - Buena interfaz gráfica: kcachegrind/qcachegrind.
- ⊙ Manual: <http://valgrind.org/docs/manual/manual.html>

- ⊙ Analizador de rendimiento estadístico basado contadores hardware y eventos.
- ⊙ Uso habitual:
 1. Recopilar datos con “operf”, “ocount” o “opgprof”.
 2. Analizar los resultados con “opreport” o “opannotate”.

- ⊙ Analizador de rendimiento estadístico basado en contadores hardware y eventos.
- ⊙ Capaz de registrar gran número de eventos: “perf list”.
- ⊙ Uso habitual:
 1. Recopilar datos con “perf record -e 'evento' -- ./exe”.
 2. Analizar los resultados con “perf report”.
- ⊙ Manuales:
 - <https://perf.wiki.kernel.org/index.php/Tutorial>
 - <http://sandsoftwaresound.net/perf/perf-tutorial-hot-spots/>

Primer ejemplo: facil.cc

facil.cc

```
const int N = 1 << 25;

int f1()
{
    int r = 0;
    for (int i = 0; i < N; ++i)
        r += i;
    return r;
}

int f2()
{
    int r = 0;
    for (int i = 0; i < 2 * N; ++i)
        r += i;
    return r;
}

int main()
{
    return f1() + f2();
}
```

- ⊙ Pruebe a analizar su rendimiento con...
 1. gprof
 2. valgrind
 3. perf
- ⊙ Este es un programa de prueba que no tiene ningún misterio pero le ayudará a empezar a practicar con los analizadores de rendimiento.
- ⊙ Si le parece difícil analizar facil.cc pruebe con facil2.cc.

ijk.cc

```
template<class T> void ijk(T a[N][N], T b[N][N], T
    c[N][N])
{
    for (size_t i = 0; i < N; ++i)
        for (size_t j = 0; j < N; ++j)
            for (size_t k = 0; k < N; ++k)
                a[i][j] += b[i][k] * c[k][j];
}
```

- ⊙ ¿Qué analizador usar valgrind o perf?
- ⊙ Creemos saber cual es la combinación óptima de índices...
- ⊙ ¿El tipo de los vectores a, b y c influye en el resultado?
 - Descubre la verdad con “make tipos” y “make ijk.perf”.
 - Los motivos están escondidos en el binario...
- ⊙ Estudie el patrón de accesos a memoria: “make ijk.cga”.

- ⊙ ¿Se pueden implementar de manera iterativa? ¿Merece la pena?
- ⊙ Compare las versiones 2 y 3 de Fibonacci... ¿Cuál es mejor y por qué?
- ⊙ ¿Qué implementación de la función de Ackermann es mejor y por qué?

- ⦿ Analice este programa de cálculo con matrices e intente mejorarlo.

- ⦿ Este ejemplo pone de manifiesto un tipo de optimización muy importante empleada por los lenguajes interpretados y del cual carecen los compilados.
- ⦿ Estudie si es rentable o no el cambio que este programa hace en sí mismo para logra una mayor eficiencia.

Ejercicios de exámenes: `axb.cc` y `struct.cc`

- ⊙ `axb.cc` y `struct.cc` son ejercicios que se ha pedido resolver en exámenes.
- ⊙ Intente escribir mejores versiones y recuerde que usted tiene la suerte de poder utilizar un ordenador.
- ⊙ GCC dispone de una opción `-fopt-info` que nos aporta información interna del compilador sobre las optimizaciones aplicadas a nuestro código fuente.
- ⊙ Una vez concluido su intento, eche un vistazo a algunas de las soluciones aportadas por profesores y alumnos:
`axb-etal.cc` y `struct-etal.cc`