Arquitectura de Sistemas

Análisis de rendimiento

Gustavo Romero López

Updated: 10 de abril de 2025

Departamento de Ingeniería de Computadores, Automática y Robótica

Objetivos

- 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 estadístico: `oprofile` y `perf`
- Análisis con gráficos de llamas (*flame graphs*)

gprof

- 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/

`valgrind`: http://valgrind.org/

- 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.
  - Excelente interfaz gráfica: `kcachegrind`/`qcachegrind`.
  - Multitud de herramientas: cachegrind, callgrind, helgrind, lackey, massif, memcheck,...
- Manual: http://valgrind.org/docs/manual/manual.html

`oprofile`: http://oprofile.sourceforge.net/news/

- 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`".

`perf`: https://perf.wiki.kernel.org/index.php/Main_Page

- 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 exe.log`".
- 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 << 28; // 256M

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;
}
```

- 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 prácticar con los analizadores de rendimiento.
- Si le parece difícil analizar `facil.cc` pruebe con `facil2.cc`.

Segundo ejemplo: leyendo desde memoria `lectura.cc`

lectura.cc

```
int f1()
{
    return std::accumulate(std::begin(v), std::end(v), 0);
}

int f2()
{
    return std::accumulate(std::rbegin(v), std::rend(v), 0);
}
```

- ¿Es capaz de adivinar qué función acaba antes?
  - `make lectura.perf`
- ¿Por qué?
  - `make lectura.cla`

Tercer ejemplo: escribiendo en memoria `escritura.cc`

`escritura.cc`

```
void f1(int i)
{
    std::ranges::fill(v, i);
}

void f2(int i)
{
    std::fill(std::rbegin(v), std::rend(v), i);
}
```

- ¿Qué diferencia de rendimiento hay entre estas funciones?
  - `make escritura.perf`
- ¿Por qué?
  - `make escritura.cla`

La tormenta perfecta: multiplicación de matrices `ijk.cc`

`ijk.h`

```
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 ijk.perf`".
  - Los motivos están escondidos en el binario...
- Estudie el patrón de accesos a memoria: "`make ijk.cla`".

Algoritmos recursivos: `fibonacci.cc` y `ackermann.cc`

- ¿Se pueden implementar de manera iterativa?
- ¿Merecen la pena las versiones paralelas?
- 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?

Algoritmos iterativos: `matrix.h` y `matrix.cc`

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

Código automodificable: `selfmod.cc`

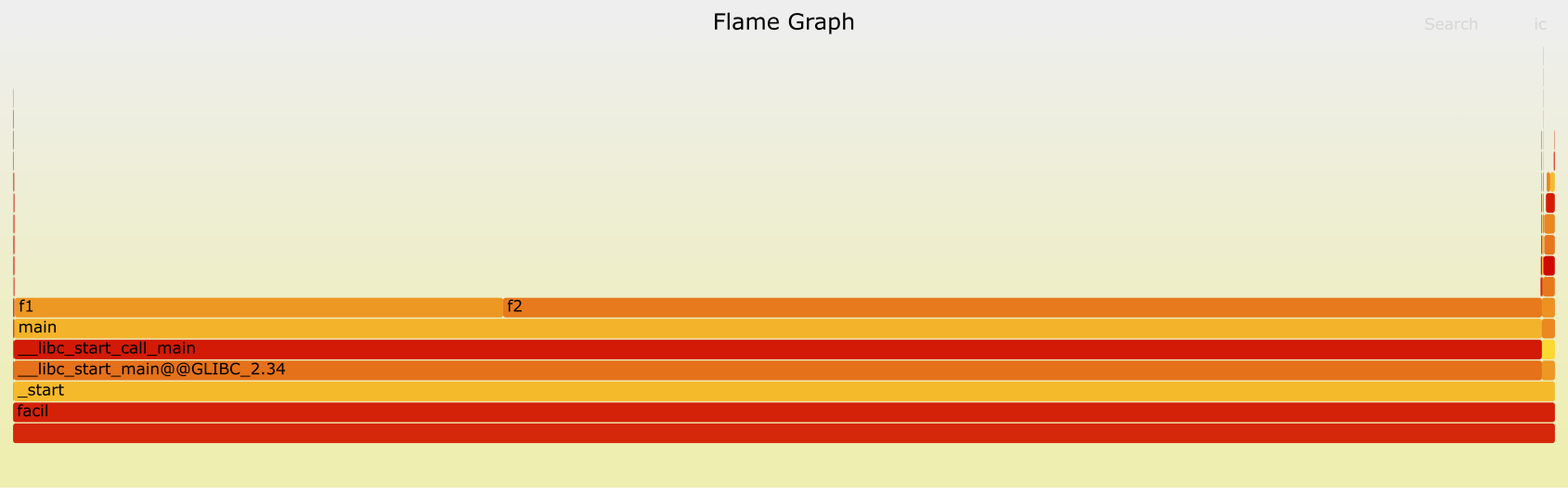
- Este ejemplo pone de manifiesto un tipo de optimización muy importante empleada por los lenguajes interpretados y del cual carecen los compilados.
- Estudie si es rentable o no el cambio que este programa hace en sí mismo para logra una mayor eficiencia.

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

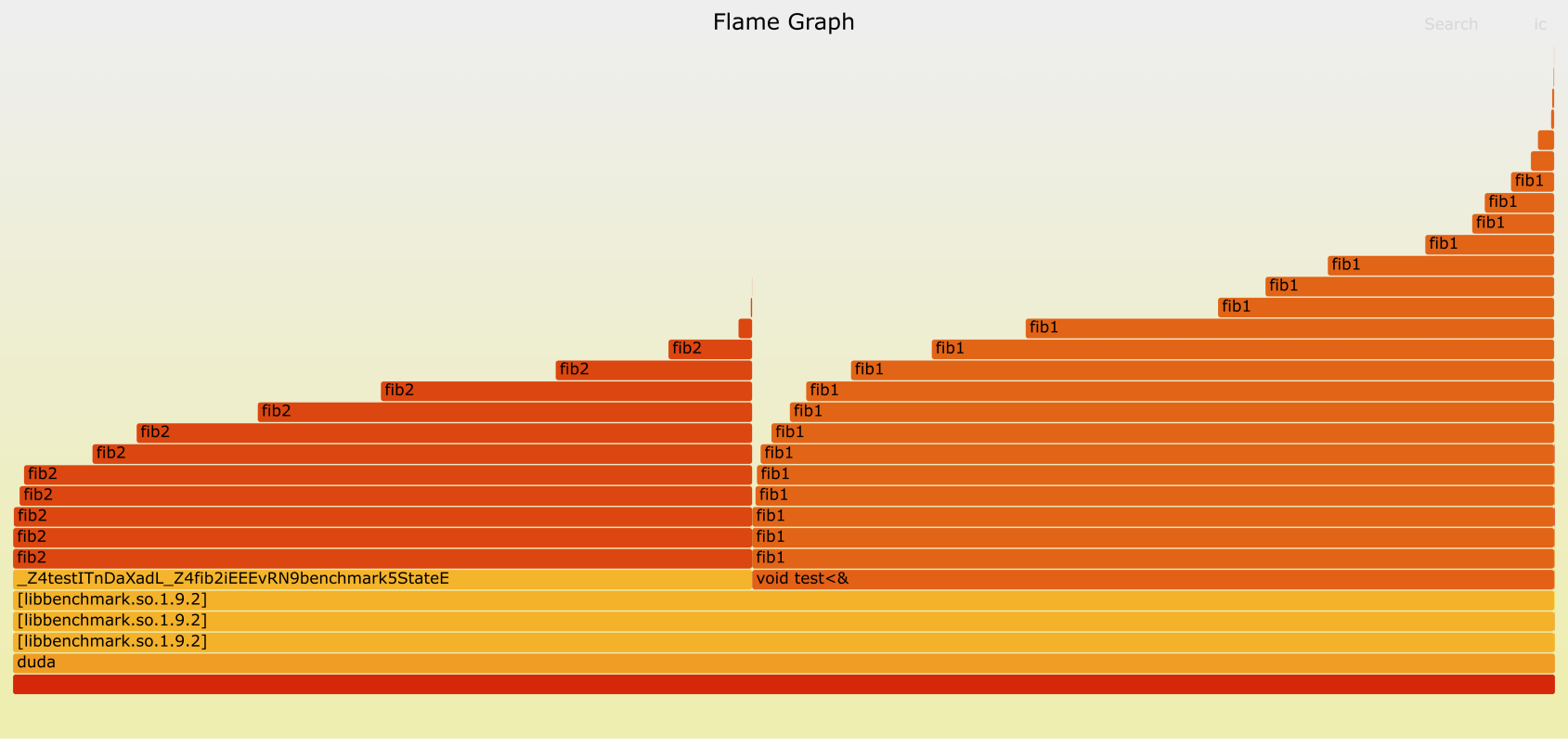
- `axb.cc` y `struct.cc` son ejercicios que se ha pedido resolver en exámenes.
- Intente escribir mejores versiones y recuerde que usted tiene la suerte de poder utilizar un ordenador.
- GCC dispone de una opción `-fopt-info` que nos aporta información interna del compilador sobre las optimizaciones 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`

Análisis mediante gráficos de llamas (*flame graphs*)

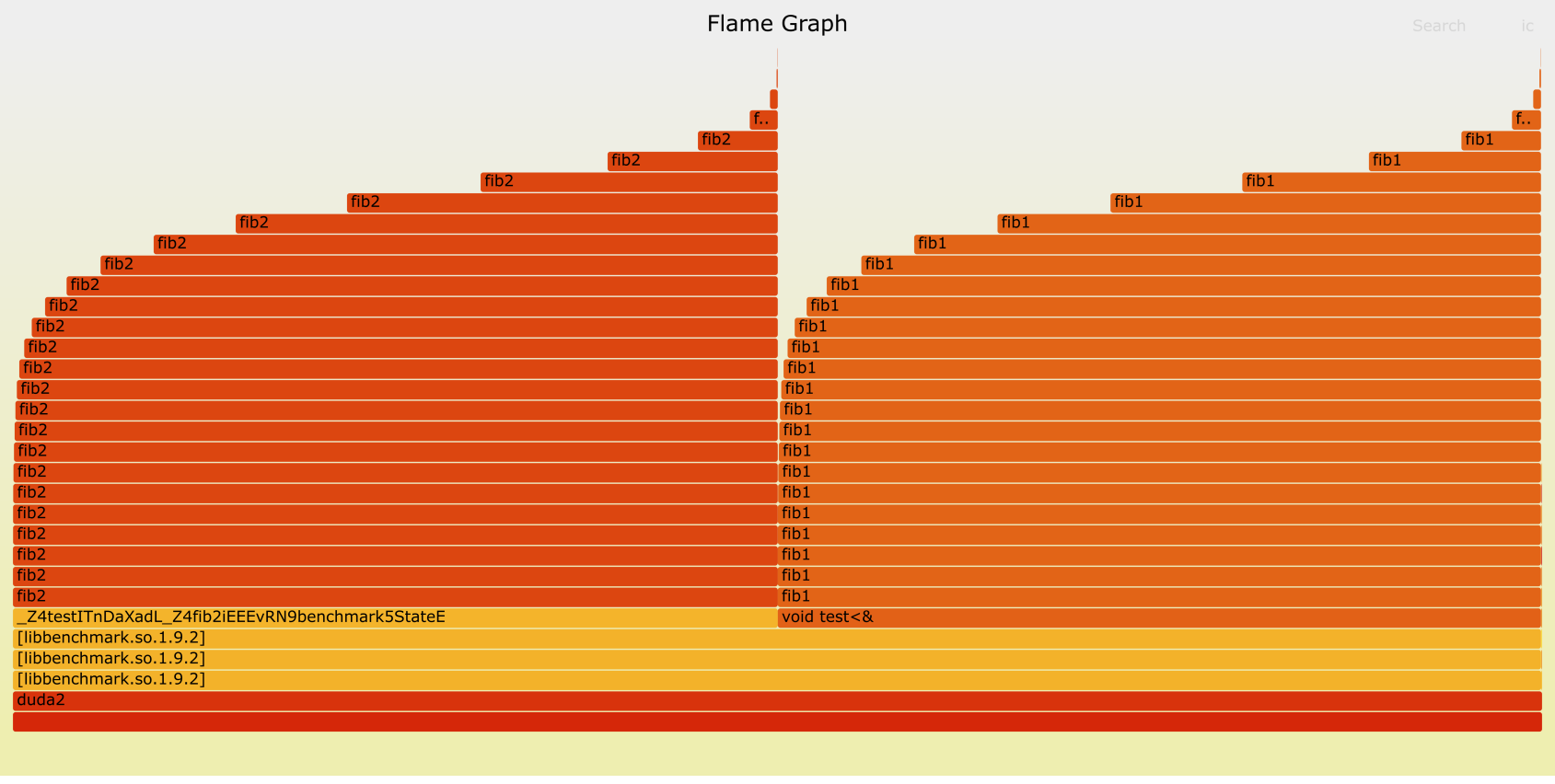
- Muestran la traza de pila de la ejecución de un programa.
- http://www.brendangregg.com/flamegraphs.html
- Interpretación:
  - Cada bloque representa una llamada a función.
  - Su ancho es proporcional al tiempo consumido.
  - Se apilan para mostrar el orden de las llamadas.



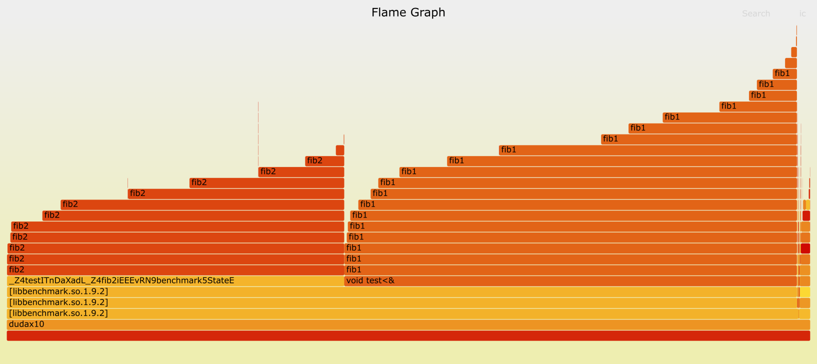
duda.svg: Google Benchmark nos induce a error



duda2.svg: “`-fno-inline/-Og`” elimina la mejora



dudax10.svg: corrección mediante “->Iterations(10)”



Tenéis otros muchos ejemplos para practicar...

- `axb-etal.cc`: Problema de cálculo propuesto en examen.
- `cache.cc`: Interferencia constructiva/destructiva de caché.
- leak.c y smart.cc: Fugas de memoria.
- `map.cc`: Vectores asociativos.
- `pila.cc`: Pilas en C y C++.
- `prime.cc`: Test de primalidad.
- `rng.cc`: Generadores de números aleatorios.
- `sort.cc`: Algoritmos de ordenación.
- `struct-etal.cc`: Problema de examen sobre estructuras.
- `vector.cc`: Suma de vectores.