Valgrind: Profiling y debugging

Intraway Corporation

IWay Tech Talks

Nicolás Brailovsky

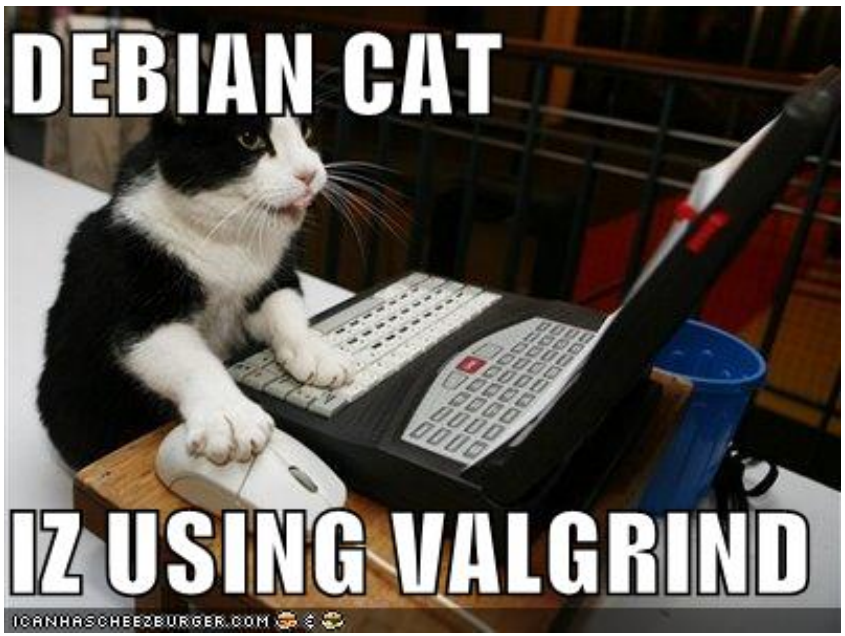
December 31, 2012

¿Qué es?

- Paquete de herramientas para analizar un programa en ejecución
  - Memoria
  - Performance
  - File descriptors
  - Race conditions
  - ...
- Un framework para desarrollar herramientas de análisis

¿Para qué utilizarlo?

- Complemento a herramientas de análisis estático
- Complemento para el gdb
- Detectar problemas "aleatorios"
- Detectar problemas que requieren largo tiempo de ejecución



¿Cómo funciona?

- Se crean wrappers para los syscall y para malloc/free
- Cada bloque del heap lleva guardas
- Simula una CPU: analiza cada instrucción
- Cada herramienta puede agregar distintas caracteristicas

¿Cómo funciona?

- Se crean wrappers para los syscall y para malloc/free
- Cada bloque del heap lleva guardas
- Simula una CPU: analiza cada instrucción
- Cada herramienta puede agregar distintas caracteristicas

¿Cómo funciona?

- Se crean wrappers para los syscall y para malloc/free
- Cada bloque del heap lleva guardas
- Simula una CPU: analiza cada instrucción
- Cada herramienta puede agregar distintas caracteristicas

Gandalf dice:

Una aplicación bajo Valgrind consume el triple de memoria y es 25 a 50 veces mas lento

Flags generales

- *–tool=<toolname>* Herramienta a utilizar
- *–log-file=<archivo>* Redirigir la salida de Valgrind
- *–trace-children* Seguir un hijo después de fork/exec
- *–track-fds* Lista los FDs en uso
- *–track-origins* Mostrar origen de valores no inicializados (V 3.4+)
- *-v* Verbose

Verificaciones comunes

Independientemente de la herramienta, Valgrind . . .

- detecta una terminación erronea (segfaults!)
- advierte si existen memory leaks
- lleva cuenta de los file descriptors (con *–track-fds*)
- es lento

Letra chica

Independientemente de la herramienta, Valgrind . . .

- no se lleva bien con las variables estáticas
- tiene un limite de threads
- es lento!

Memcheck: Introducción

- Herramienta por default
- Busca memory leaks
- Detecta la utilización de variables no inicializadas
- Advierte buffer overruns y problemas similares

Memcheck: Off by one

```
{
  char *msg = new char[4];
  strcpy(msg, "hola");
}
```

Error en valgrind:

```
Invalid write  of size 1
  at 0x4024BD7: memcpy (mc_replace_strmem.c:402)
  by 0x80484DE: main  (main.cpp:5)
Address 0x42ae02c is  0 bytes after a block of size 4 alloc'd
  at 0x4022F14: operator new[](unsigned) (vg_replace_malloc.c:268)
  by 0x80484C0: main  (main.cpp:4)
```

Memcheck: Memory leak

Ejecutando el programa con –leack-check=full

```
{
  char *msg = new char[4];
  strcpy(msg, "hola");
}
```

Error en valgrind:

```
5 bytes in 1 blocks are definitely lost in loss record 1 of 1
   at 0x4022F14: operator new[](unsigned) (vg_replace_malloc.c:268)
   by 0x80484C0: main (main.cpp:4)

LEAK SUMMARY:
    definitely lost: 5 bytes in 1 blocks.
     possibly lost: 0 bytes in 0 blocks.
   still reachable: 0 bytes in 0 blocks.
        suppressed: 0 bytes in 0 blocks.
```

Memcheck: Free/delete

```
{
  char *msg = new char[5];
  strcpy(msg, "hola");
  free(msg);
}
```

Error en valgrind:

```
Mismatched free() / delete / delete []
   at 0x402265C: free (vg_replace_malloc.c:323)
   by 0x8048519: main (main.cpp:7)
 Address 0x42ae028 is 0 bytes inside a block of size 5 alloc'd
   at 0x4022F14: operator new[](unsigned) (vg_replace_malloc.c:268)
   by 0x80484F0: main (main.cpp:5)
```

Memcheck: Uninitialised value / SIGSEGV

```
{
  char *msg;
  strcpy(msg, "hola");
  free(msg);
}
```

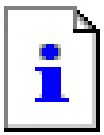
Error en valgrind:

```
Conditional jump or move depends on uninitialised value(s)
  at 0x4024AC5: memcpy (mc_replace_strmem.c:77)
  by 0x804846F: main  (main.cpp:6)

Conditional jump or move depends on uninitialised value(s)
  at 0x4024B7C: memcpy (mc_replace_strmem.c:80)
  by 0x804846F: main  (main.cpp:6)
...
Process terminating with default action of signal 11 (SIGSEGV)

Bad permissions for mapped region at address 0x400DDB0
  at 0x4024BA8: memcpy (mc_replace_strmem.c:402)
  by 0x804846F:  main (main.cpp:6)
```

Memcheck: Variable estática

```
char msg[4];
int main() {
  strcpy(msg, "hola");
  return 0;
}
```

Error en valgrind:



The page cannot be found

The page you are looking for might have been removed, had its name changed, or is temporarily unavailable.

Memcheck: conclusión

- Una de las herramientas mas prácticas
- Detecta el error y su origen
- Sencilla de utilizar

Para tener en cuenta

- Requiere compilar con -g (para ver la linea del error)
- No se lleva bien con -On: las optimizaciones generan falsos positivos
- No controla variables estáticas

Callgrind: introducción

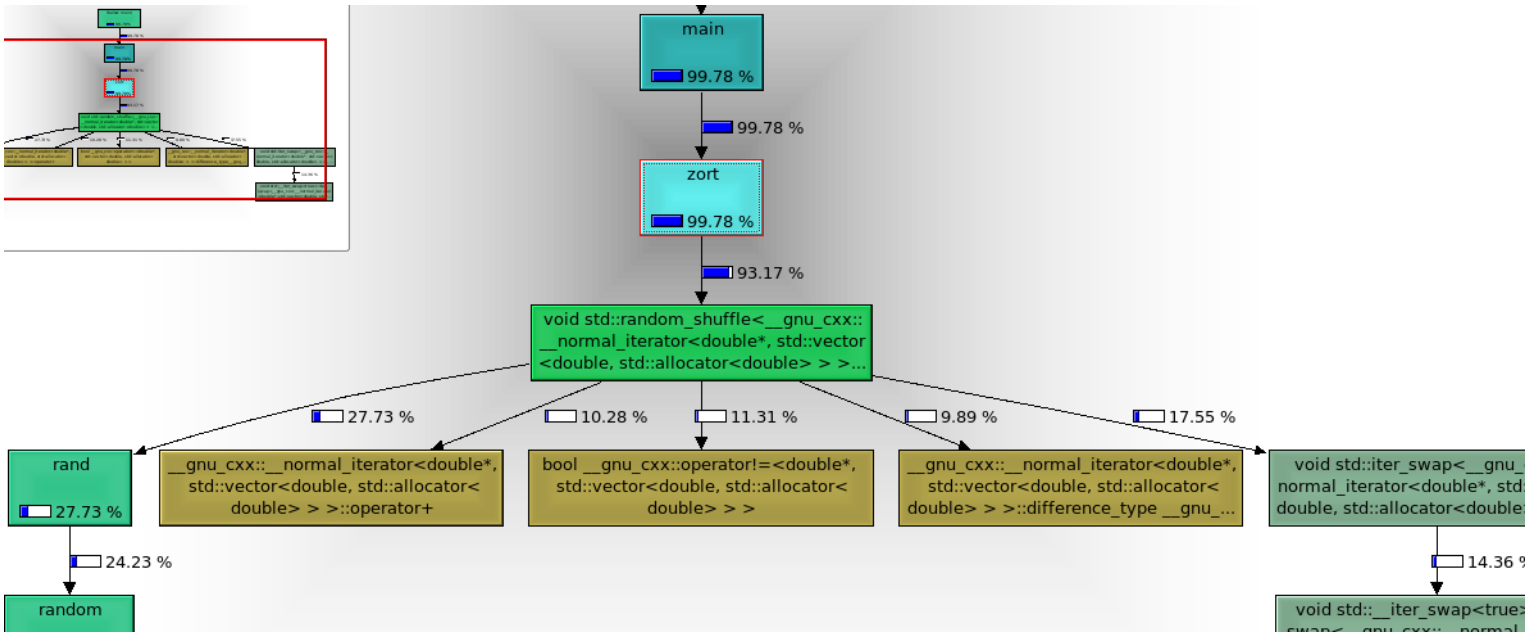
- Extensión de cachegrind
- Profiler para cache L1 y L2
- Crea un profile de cada función utilizada
- Permite generar gráficos de llamada a partir de stacktraces

Para tener en cuenta . . .

- es necesario utilizar KCacheGrind para interpretar los resultados

Callgrind: ejemplo

Gráfico de llamadas generado con KCacheGrind



Callgrind: ejemplo

Utilización del cache

```
--18822-- LRU Contxt Misses: 1672
--18822-- LRU BBCC Misses:   16
--18822-- LRU JCC Misses:    384
--18822-- BBs Executed:      717200001
--18822-- Calls:             281943034
--18822-- CondJMP followed:  0
--18822-- Boring JMPs:       0
--18822-- Recursive calls:   6
--18822-- Returns:           281943034
```

RTFM

Callgrind: ejemplo

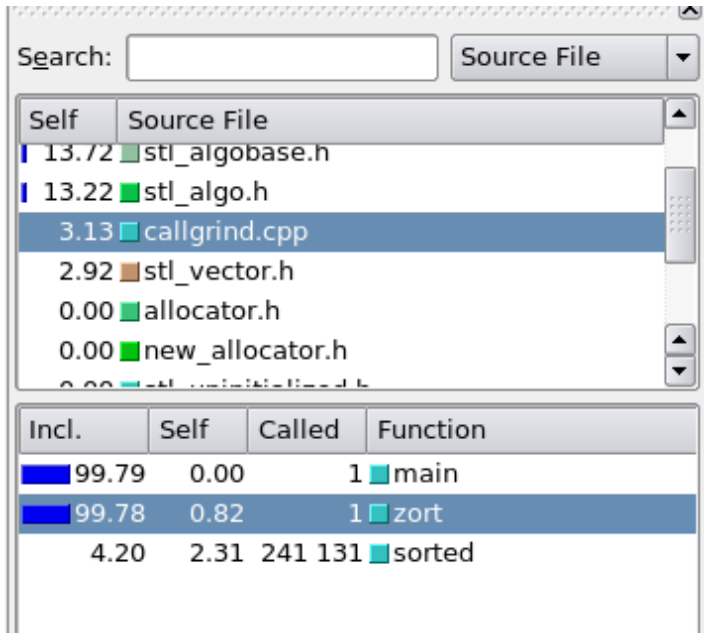
Utilización del cache

```
--18822-- LRU Contxt Misses: 1672
--18822-- LRU BBCC Misses:   16
--18822-- LRU JCC Misses:    384
--18822-- BBs Executed:      717200001
--18822-- Calls:             281943034
--18822-- CondJMP followed:  0
--18822-- Boring JMPs:       0
--18822-- Recursive calls:   6
--18822-- Returns:           281943034
```

RTFM

Callgrind: ejemplo

Profiling de una aplicación (KCacheGrind)



¿Qué estará haciendo zort()?

Callgrind: Ejemplo

```
void zort(vector<double> &data) {
  while (!sorted(data))
    random_shuffle(data.begin(), data.end());
}
```

Parece que bogosort no es un buen algoritmo...

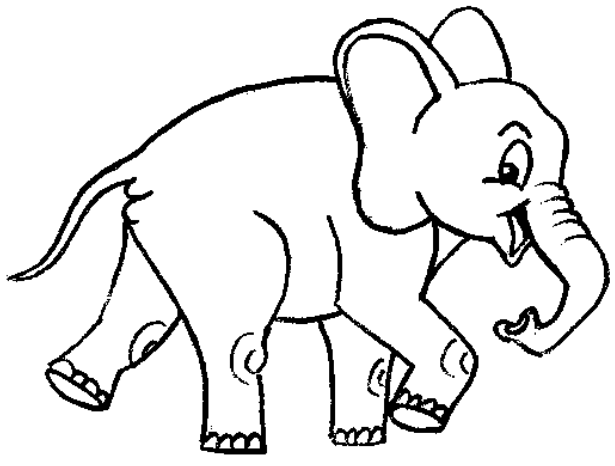
Callgrind: conclusión

- Muestra que métodos son hojas en el callgraph
- En una aplicación real callgrind encuentra bottlenecks

Massif

Massif sirve como . . .

- Memory profiler



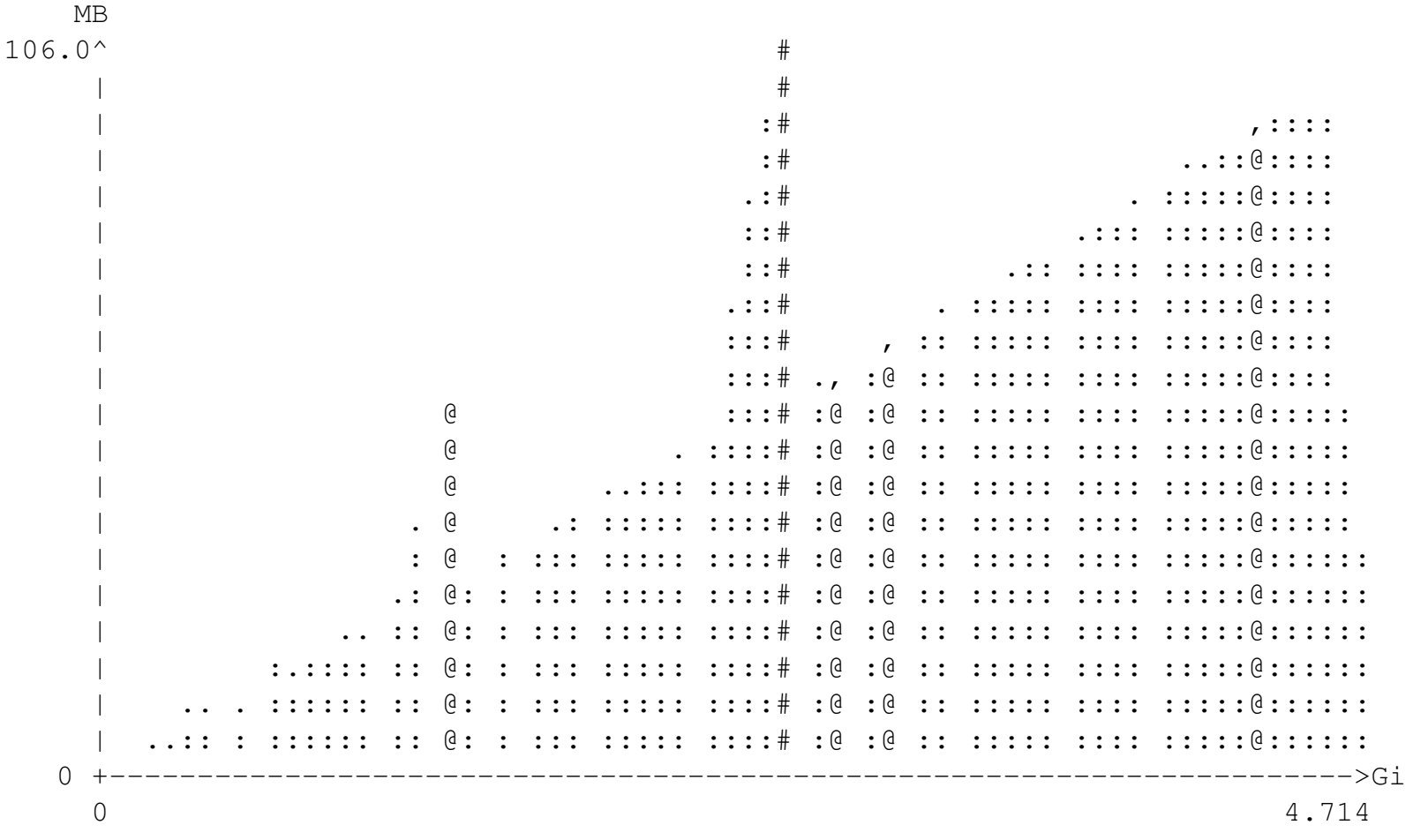
Massif: Ejemplo

Implementemos este cambio para ver si Callgrind funciona bien

```
long  zort(vector<double> &data) {
 vector< vector<double> > intentos;

while (!sorted(data)) {
  std::random_shuffle(data.begin(), data.end());
   intentos.push_back(vector<double>(data));
}

 return intentos.size();
}
```

Massif: Ejemplo

Funciona, bogosort parece ser el culpable, pero ¿que pasó con la memoria?

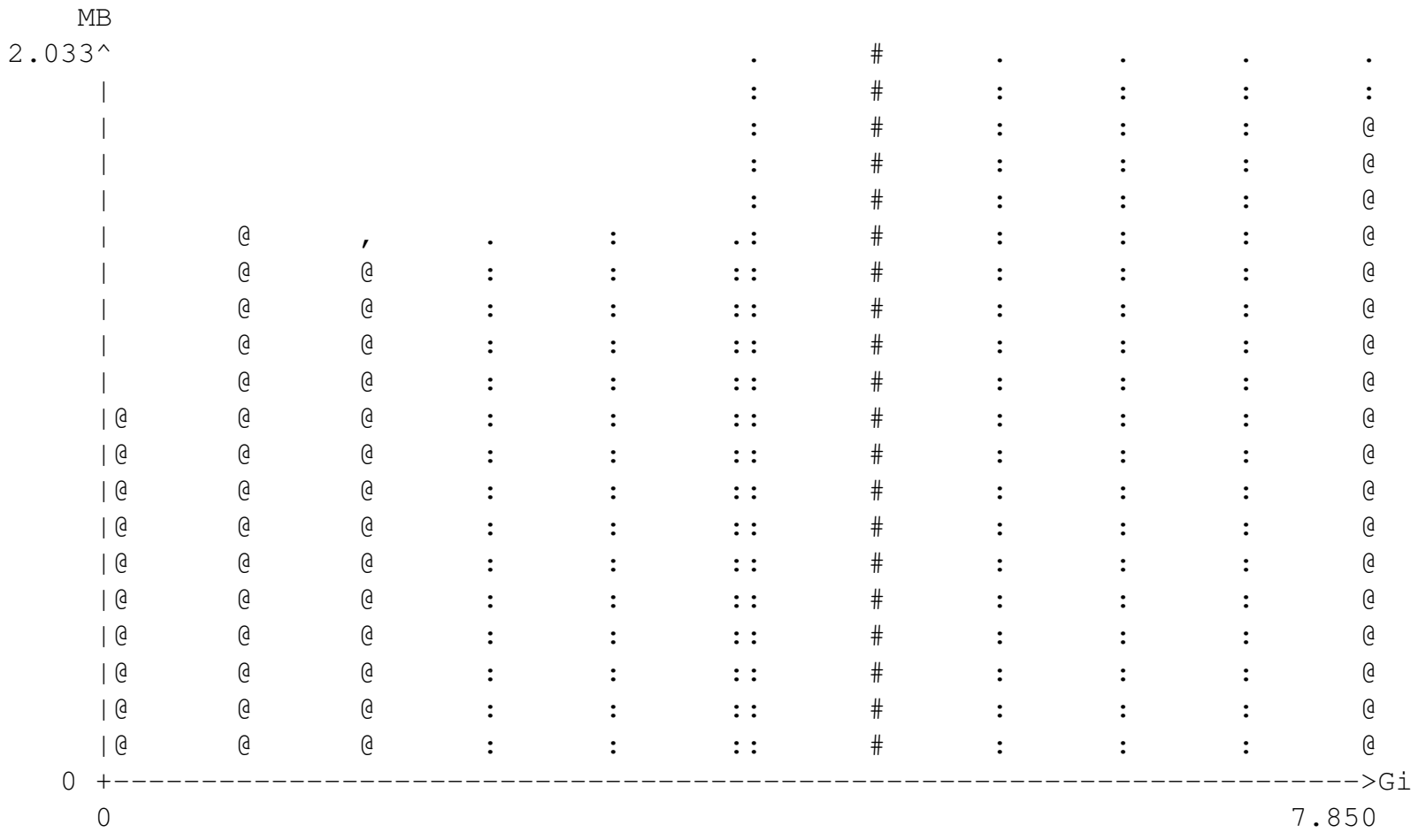


Massif: Ejemplo

De los 106 MB, un 92% corresponde a vector, reservado en massif.cpp:29

```
92.45% (6,422,768B) (heap allocation functions) malloc/new/new[], --alloc-fns, etc.
->75.47% (5,243,120B) 0x8049561: __gnu_cxx::new_allocator<double>::allocate(unsigned, [...]
| ->75.47% (5,243,120B) 0x8049585: std::_Vector_base<double, std::allocator<double> [...]
[...]
|
->16.98% (1,179,648B) 0x804963B: __gnu_cxx::new_allocator<std::vector<double, [...]
    [...]
        ->16.98% (1,179,648B) 0x8048C19: zort(std::vector<double, std::allocator<double> >&)
          ->16.98% (1,179,648B) 0x8048D48: main (massif.cpp:29)
```

Massif: Un ejemplo mas completo



Massif: Un ejemplo mas completo

Memoria utilizada

```
99.48% (1,605,440B) (heap allocation functions) malloc/new/new, -alloc-fns, etc.
->65.48% (1,056,768B) 0x80497B5: math::matrix<double>::base_mat::base_mat(unsigned, unsigned, double**) (in va
| ->64.97% (1,048,576B) 0x8049834: math::matrix<double>::matrix(unsigned, unsigned) (in valgrind/mkv3)
| | ->64.97% (1,048,576B) 0x804A475: math::matrix<double>::operator*=(math::matrix<double> const&) (in valgrin
| | | ->64.97% (1,048,576B) 0x804A6CD: Mkv::get_tr_matrix() (in valgrind/mkv3)
| | |   ->32.49% (524,288B) 0x804A864: Mkv::compare_to(Mkv&) (in valgrind/mkv3)
| | |   | ->32.49% (524,288B) 0x8049295: main (in valgrind/mkv3)
| | |   |
| | |    ->32.49% (524,288B) 0x804A879: Mkv::compare_to(Mkv&) (in valgrind/mkv3)
| | |     ->32.49% (524,288B) 0x8049295: main (in valgrind/mkv3)
| | |
| | ->00.00% (0B) in 1+ places, all below ms_print's threshold (01.00%)
| |
| ->00.51% (8,192B) in 1+ places, all below ms_print's threshold (01.00%)
|
->32.49% (524,288B) 0x804960D: math::matrix<long>::base_mat::base_mat(unsigned, unsigned, long**) (in valgrind
| ->32.49% (524,288B) 0x804968C: math::matrix<long>::matrix(unsigned, unsigned) (in valgrind/mkv3)
|   ->32.49% (524,288B) 0x8049AFE: Mkv::Mkv<unsigned char>(Tokenizer<unsigned char>&) (in valgrind/mkv3)

|     ->16.24% (262,144B) 0x8049239: main (in valgrind/mkv3)
|     |
|     ->16.24% (262,144B) 0x804927D: main (in valgrind/mkv3)
|
->01.02% (16,384B) 0x40937EB: std::basic_filebuf<char, std::char_traits<char> >::_M_allocate_internal_buffer()
| ->01.02% (16,384B) 0x4097231: std::basic_filebuf<char, std::char_traits<char> >::open(char const*, std::_Ios
|   ->01.02% (16,384B) 0x4097C0E: std::basic_ifstream<char, std::char_traits<char> >::basic_ifstream(char cons
|     ->01.02% (16,384B) 0x8049499: FileTokenizer<unsigned char>::FileTokenizer(char const*) (in valgrind/mkv3
|       ->01.02% (16,384B) 0x80494B3: CharTokenizer::CharTokenizer(char const*) (in valgrind/mkv3)

|        ->01.02% (16,384B) in 2 places, all below massif's threshold (01.00%)
|
->00.50% (8,000B) in 1+ places, all below ms_print's threshold (01.00%)
```

Massif: conclusión

- Herramienta complementaria a Callgrind
- Permite detectar bottlenecks
- Los cambios en la última versión no generan postscript
- El formato actual es horrible!

Helgrind: Introducción

Helgrind muestra . . .

- condiciones de carrera
- deadklocks

Helgrind: Ejemplo

Programa de ejemplo:

```
class Hormiguita : public ACE_Task_Base {
  public:
  int &num;
  Hormiguita(int &num) : num(num) {
    activate(..., 10);
  }

  int svc() {
     while (num < 1000) num++;
    return 0;
  }
};
```

Helgrind: Ejemplo

```
==2404== Possible data race during read of size 4 at 0xbee16e78 by thread #3
==2404==    at 0x8049D4D: Worker::svc() (helgrind.cpp:20)
==2404==    by 0x413D551: ACE_Task_Base::svc_run(void*) (Task.cpp:275)
==2404==    by 0x413E287: ACE_Thread_Adapter::invoke_i() (Thread_Adapter.cpp:149)
==2404==    by 0x413E465: ACE_Thread_Adapter::invoke() (Thread_Adapter.cpp:98)
==2404==    by 0x40CC2D0: ace_thread_adapter (Base_Thread_Adapter.cpp:124)
==2404==    by 0x4026BFE: mythread_wrapper (hg_intercepts.c:194)
==2404==    by 0x441F4FA: start_thread (in /lib/tls/i686/cmov/libpthread-2.7.so)
==2404==    by 0x43A1E5D: clone (in /lib/tls/i686/cmov/libc-2.7.so)
==2404==  This conflicts with a previous write of size 4 by thread #2
==2404==    at 0x8049D44: Worker::svc() (helgrind.cpp:20)
==2404==    by 0x413D551: ACE_Task_Base::svc_run(void*) (Task.cpp:275)
==2404==    by 0x413E287: ACE_Thread_Adapter::invoke_i() (Thread_Adapter.cpp:149)
==2404==    by 0x413E465: ACE_Thread_Adapter::invoke() (Thread_Adapter.cpp:98)
==2404==    by 0x40CC2D0: ace_thread_adapter (Base_Thread_Adapter.cpp:124)
==2404==    by 0x4026BFE: mythread_wrapper (hg_intercepts.c:194)
==2404==    by 0x441F4FA: start_thread (in /lib/tls/i686/cmov/libpthread-2.7.so)
==2404==    by 0x43A1E5D: clone (in /lib/tls/i686/cmov/libc-2.7.so)
```

Helgrind: conclusión

- Una excelente idea, pero . . .
- el S/N ratio es muy bajo, mas aún en ACE [1]

[1] En el slide anterior solo se muestra una parte del log

Further reading I

RTFM

man valgrind

2000-2009 Valgrind Developers

Valgrind website

http://valgrind.org