the HOARD memory allocator

by Emery Berger
emery@cs.umass.edu
www.cs.mass.edu/~emery
Hoard home page: www.hoard.org

TABLE OF CONTENTS

hoard: *(v.) To amass and put away (anything valuable) for preservation, security, or future use; to treasure up: esp. money or wealth.*
(Oxford English Dictionary)

Introduction

The Hoard memory allocator is a fast, scalable, and memory-efficient memory allocator for shared-memory multiprocessors. It runs on a variety of platforms, including Linux, Solaris, and Windows.

Why Hoard?

Contention

Multithreaded programs often do not scale because the heap is a bottleneck. When multiple threads simultaneously allocate or deallocate memory from the allocator, the allocator will serialize them. Programs making intensive use of the allocator actually slow down as the number of processors increases. Your program may be allocation-intensive without you realizing it, for instance, if your program makes many calls to the C++ Standard Template Library (STL).

False Sharing

The allocator can cause other problems for multithreaded code. First, it can lead to *false sharing* in your application: threads on different CPUs can end up with memory in the same cache line, or chunk of memory. Accessing shared cache lines can be hundreds of times slower than accessing unshared cache lines.

Blowup

Multithreaded programs can also lead the allocator to *blowup* memory consumption. This effect can multiply the amount of memory needed to run your application by the number of CPUs on your machine: four CPUs could mean that you need four times as much memory.

Hoard is a fast allocator that solves all of these problems.

How Do I Use Hoard?

Hoard is a drop-in replacement for `malloc()`, etc. In general, you just link it in or set just one environment variable. You do not have to change your source code in any way. See the section "Windows Builds" below for more information for particular platforms.

Who's Using Hoard?

Users of Hoard include AOL, British Telecom, Crystal Decisions, Entrust, Novell, Coyote Systems (for their BEMEngine product), OpenWave Systems (for their Typhoon & Twister servers), and Reuters. Open source projects using Hoard include Ardour, the Bayonne GNU telephony server and the GNU Common C++ system.

Building Hoard

You can use the available pre-built binaries or build Hoard yourself. Hoard is written to work on Windows and any variant of UNIX that supports threads, and should compile out of the box. Rather than using Makefiles or configure scripts, Hoard includes custom scripts that all start with the prefix `compile`.

Linux & Solaris Builds

You can compile Hoard out of the box for Linux and Solaris *using the GNU compilers only*.

```
./compile-hoard
```

Windows Builds

There are now two alternative ways of using Hoard with Windows. The first approach builds a DLL, `libhoard.dll` and its associated library `libhoard.lib`.

```
.\compile-dll
```

The second approach relies on Microsoft Research's Detours (http://research.microsoft.com/sn/detours). With Detours, you can take advantage of Hoard without having to relink your applications. Install Detours into `C:\detours`, and then build the Hoard detours library:

```
.\compile-detours
```

Using Hoard

UNIX

In UNIX, you can use the `LD_PRELOAD` variable to use Hoard instead of

the system allocator for any program not linked with the `-static` option (that's most programs). Below are settings for Linux and Solaris.

Solaris

```
setenv LD_PRELOAD \
"/path/to/libhoard.so:/usr/lib/libthread.so \
:/usr/lib/librt.so:/usr/lib/libCrun.so.1"
```

Note: For some security-sensitive applications, Solaris requires you place libraries used in `LD_PRELOAD` into the `/usr/lib/secure` directory. In that event, after copying these libraries into `/usr/lib/secure`, set `LD_PRELOAD` as below:

```
setenv LD_PRELOAD "libhoard.so:libthread.so"
```

Linux

```
setenv LD_PRELOAD "/path/to/libhoard.so:/usr/lib/libdl.so"
```

A Debian package for Hoard is available at packages.debian.org/libhoard.

Windows

You can now use Hoard in two ways on Windows.

Using libhoard

When you build Hoard under Windows, you will get two files: `libhoard.dll` and `libhoard.lib`. Put the following into your source code as the very first lines:

```
#if defined(USE_HOARD) && defined(_WIN32)
#pragma comment(lib, "libhoard.lib")
#endif
```

The best approach is to put this stanza into the first part of a header file included by all of your code. The `pragma` ensures that Hoard loads before any other library (you will need `libhoard.lib` in your path). When you execute your program, as long as `libhoard.dll` is in your path, your program will run with Hoard instead of the system allocator.

Note that you **must** compile your program with the /MD flag, as in:

```
cl /MD /G6 /Ox /DUSE_HOARD=1 myprogram.cpp
```

Hoard will not work if you use another switch (like `/MT`) to compile your program.

Using Detours

By using Detours, you can take advantage of Hoard's benefits without relinking your Windows application (as long as it is dynamically linked to the C runtime libraries). With this approach, Hoard performs memory allocation operations for small objects (< 8K) in your application. Larger objects will continue to be managed via the Windows allocator: the original allocation instructions will be executed (i.e., they are not bypassed).

You will need to use one of two of the Detours tools (`setdll.exe` or `withdll.exe`) in conjunction with this version of Hoard. To temporarily use Hoard as your allocator, use `withdll`:

```
withdll –d:hoarddetours.dll myprogram.exe
```

If you want your program to use Hoard without having to invoke `withdll` every time, you can use `setdll` to add it to your executable:

```
setdll –d:hoarddetours.dll myprogram.exe myprogram.exe
```

You can always remove Hoard from your executable by using the `–r` option of `setdll`.

License Information