CHAPTER 15

# Improving Performance Through Build Options

It's important how you build mod_perl-enabled Apache. The build process influences the size of the *httpd* executable—for example, some irrelevant modules might slow down performance.

When you build Apache, it strips the debug symbols by default, so you don't have to strip them yourself. For production use, you definitely shouldn't build mod_perl with debugging options enabled. Apache and mod_perl do not add these options unless you explicitly require them. In Chapter 21 we talk about debug build options in detail.

## Server Size as a Function of Compiled-in Features

You might wonder if it's better to compile in only the required modules and mod_perl hooks, or if it doesn't really matter. To answer this question, let's first make a few compilations and compare the results.

We'll build mod_perl starting with:

```
panic% perl Makefile.PL APACHE_SRC=../apache_1.3.x/src \
       DO_HTTPD=1 USE_APACI=1
```

and followed by one of these option groups, in turn:

- Default (no arguments)
- Minimum:

```
APACI_ARGS='--disable-module=env, \
            --disable-module=negotiation, \
            --disable-module=status, \
            --disable-module=info, \
            --disable-module=include, \
            --disable-module=autoindex, \
            --disable-module=dir, \
            --disable-module=cgi, \
```

```
                --disable-module=asis, \
                --disable-module=imap, \
                --disable-module=userdir, \
                --disable-module=access, \
                --disable-module=auth'
```

- mod_perl's EVERYTHING:

```
EVERYTHING=1
```

- mod_perl's EVERYTHING and debug:

```
EVERYTHING=1 PERL_DEBUG=1
```

After recompiling with the arguments of each of these groups in turn, we can summarize the results as follows:

```
Build group    httpd size (bytes)  Difference
-----------------------------------------------
Minimum              892928         +     0
Default              994316         +101388
Everything          1044432         +151504
Everything+Debug    1162100         +269172
```

Clearly when you strip most of the defaults, the server size is slimmer. But the savings become insignificant, because you don't multiply the added size by the number of child processes if your OS supports memory sharing. The parent process is a little bigger, but it shares these memory pages with its child processes. Of course, not all the memory will be shared, but most of it will.

This is just an example to show the maximum possible difference in size. You can't actually strip everything away, because there will be Apache modules and mod_perl options that you won't be able to work without. But as a good system administrator's rule says: "Run the absolute minimum of the applications. If you don't know or need something, disable it." Following this rule to decide on the required Apache components and disabling the unneeded default components makes you a better Apache administrator.

## mod_status and ExtendedStatus On

If you build in mod_status and you also set:

```
ExtendedStatus On
```

in *httpd.conf*, on every request Apache will perform two calls to `gettimeofday(2)` (or `times(2)`, depending on your operating system). This is done so that the status report contains timing information. For highest performance, set `ExtendedStatus Off` (which is the default).

## DYNAMIC_MODULE_LIMIT Apache Build Option

If you have no intention of using dynamically loaded modules (you probably don't if you're tuning your server for every last ounce of performance), you should add *-DDYNAMIC_MODULE_LIMIT=0* when building the server. This will save RAM that's allocated only for supporting dynamically loaded modules.

## Perl Build Options

The Perl interpreter is the brain of the mod_perl server. If you can optimize Perl into doing things faster under mod_perl, you'll make the whole server faster. Generally, optimizing the Perl interpreter means enabling or disabling some build options. Let's look at a few important ones. (Note that you have to build Perl *before* you build mod_perl-enabled Apache. If you have rebuilt the Perl interpreter, make sure to rebuild mod_perl as well, or the changes won't affect mod_perl.)

You can pass build options to Perl via the *Configure* script. To specify additional C compiler flags, use the *-Accflags=...* *Configure* command-line option (e.g., *-Accflags=-DFOO* will define the C preprocessor symbol `FOO`.) You can also pass additional optimizer/debugger flags via *-Doptimize=...* (e.g., *-Doptimize='-O2 -march=pentium'*).

Don't enable Perl's thread support unless you need it, because some internal data structures are modified and/or extended under *ithreads/5005threads*—this may make certain things slower and could lead to extra memory usage.

You have a choice of using the native or Perl's own `malloc()` implementation. The default choice depends on your operating system. On some OSes the native implementation might be worse than Perl's. Unless you know which of the two is better on yours, try both and compare the benchmarks.

To build without Perl's `malloc()`, you can use the *Configure* command:

```
panic% sh Configure -Uusemymalloc
```

Note that:

```
-U == undefine usemymalloc (== use system malloc)
-D == define   usemymalloc (== use Perl's malloc)
```

The Linux OS still defaults to system `malloc()`, so you might want to configure Perl with `-Dusemymalloc`. Perl's `malloc()` is not much of an imporovement under Linux (it's about a 5–10% speed improvement according to Scott Thomason, as explained at *http://www.mlug.net/mlug-list/2000/msg00701.html*), but it makes a huge difference under Solaris (when using Sun's C compiler). Be sure also to check the *README.** file corresponding to your OS in the Perl source code distribution for specific instructions and caveats.

## Architecture-Specific Compile Options

When you build Apache and Perl, you can optimize the compiled applications to take advantage of the benefits of your machine's architecture.

Everything depends on the kind of compiler that you use, the kind of CPU(s) you use, and your OS.

For example, if you use *gcc(1)*, you might want to use *-march=pentium* if you have a Pentium CPU, or *-march=pentiumpro* for PentiumPro and above.

*-fomit-frame-pointer* makes an extra register available but disables debugging. You can also try these options, which have been reported to improve performance: *-ffast-math*, *-malign-double*, *-funroll-all-loops*, *-fno-rtti*, and *-fno-exceptions*. See the *gcc(1)* manpage for details about these.

You may also want to change the default `-O2` flag to a flag with a higher number, such as `-O3`. `-OX` (where `X` is a number between 1 and 6) defines a collection of various optimization flags; the higher the number, the more flags are bundled. The *gcc* manpage will tell you what flags are used for each number. Test your applications thoroughly (and run the Perl test suite!) when you change the default optimization flags, especially when you go beyond `-O2`. It's possible that the optimization will make the code work incorrectly and/or cause segmentation faults.

See your preferred compiler's manpage and the resources listed in the next section for detailed information about optimization.

## References

- The GCC manual: *http://gcc.gnu.org/onlinedocs/*
- "Code Optimization Using the GNU C Compiler," by Rahul U Joshi: *http://www.linuxgazette.com/issue71/joshi.html*

  This article describes some of the code optimization techniques used by the GNU C Compiler, in order to give the reader a feel of what code optimization is and how it can increase the efficiency of the generated object code.
- *Using and Porting GNU CC for Version 2.8*, by Richard Stallman (Free Software Foundation). Also available online from *http://www.delorie.com/gnu/docs/gcc/gcc_toc.html* and many other locations.
- Chapter 6 of the online book *Securing and Optimizing Linux, RedHat Edition: A Hands on Guide* talks extensively about compiler flags. It is located at *http://www.linuxdoc.org/LDP/solrhe/Securing-Optimizing-Linux-RH-Edition-v1.3/gen-optim.html*. The whole book (available in different formats) can be found at *http://www.linuxdoc.org/guides.html#securing_linux*.
- More Apache and platform-specific performance-tuning notes can be found at *http://httpd.apache.org/docs/misc/perf-tuning.html*.