Using HPCTOOLKIT with Statically Linked Programs

The HPCTOOLKIT Team

April 2, 2009

Abstract

HPCTOOLKIT is an integrated suite of tools that supports measurement, analysis, attribution and presentation of application performance for sequential and parallel programs. This document describes how to link and run a statically linked application so that you can use HPCTOOLKITś monitoring capabilities.

1 Introduction

On modern Linux systems, dynamically linked executables are the default. With dynamically linked executables, HPCTOOLKIT's `hpcrun` script uses library preloading to inject HPCTOOLKIT's monitoring code into an application's address space. However, in some cases, one wants or needs to build a statically linked executable.

- One might want to build a statically linked executable because they are generally faster if the executable spends a significant amount of time calling functions in libraries.
- On scalable parallel systems such as a Blue Gene/P or a Cray XT, at present the compute node kernels don't support using dynamically linked executables; for these systems, one needs to build a statically linked executable.

For statically linked executables, preloading HPCTOOLKIT's monitoring code into an application's address space at program launch is not an option. Instead, monitoring code must be added at link time; HPCTOOLKIT's `hpclink` script is used for this purpose.

2 Linking with `hpclink`

Adding HPCTOOLKIT's monitoring code into a statically-linked application is easy. This does not require any source-code modifications, but it does involve a small change to your build procedure. You continue to compile all of your object (`.o`) files exactly as before, but you will need to modify your final link step to use `hpclink` to add HPCTOOLKIT's monitoring code to your executable.

In your build scripts, locate the last step in the build, namely, the command that produces the final statically linked binary. Edit that command line to add the `hpclink` command at the front.

For example, suppose that the name of your executable is `myprog` and the last step in your `Makefile` links various object files and libraries as follows into a statically linked executable:

```
mpicc -o myprog -static file.o ... -l<lib> ...
```

To build a version of your executable with HPCTOOLKIT's monitoring code linked in, you would use the following command line:

```
hpclink mpicc -o myprog -static file.o ... -l<lib> ...
```

In practice, you may want to edit your `Makefile` to always build two versions of your program, perhaps naming them `myprog` and `myprog.hpc`.

3 Running a Statically Linked Binary

For dynamically linked executables, the `hpcrun` script sets environment variables to pass information to the HPCTOOLKIT monitoring library. On standard Linux systems, statically-linked executables can still be launched with `hpcrun`.

On Cray XT and Blue Gene/P systems, the `hpcrun` script is not applicable because of differences in application launch procedures. On these systems, you will need to use the `CSPROF_OPT_EVENT` environment variable to pass a list of events to HPCTOOLKIT's monitoring code, which was linked into your executable using `hpclink`. Typically, you would set `CSPROF_OPT_EVENT` in your launch script.

The `CSPROF_OPT_EVENT` environment variable should be set to a space-separated list of `EVENT@COUNT` pairs. For example, in a PBS script for a Cray XT system, you might write the following in Bourne shell or bash syntax:

```
#!/bin/sh
#PBS -l size=64
#PBS -l walltime=01:00:00
cd $PBS_O_WORKDIR
export CSPROF_OPT_EVENT="PAPI_TOT_CYC@4000000 PAPI_L2_TCM@400000"
aprun -n 64 ./myprog arg ...
```

Using the Cobalt job launcher on Argonne National Laboratory's Blue Gene/P system, you would use the `--env` option to pass environment variables. For example, you might submit a job with:

```
qsub -t 60 -n 64 --env CSPROF_OPT_EVENT="PAPI_TOT_CYC@4000000" \
    /path/to/myprog arg ...
```

4 Troubleshooting

With some compilers you need to disable interprocedural optimization to use `hpclink`. To instrument your statically linked executable at link time, `hpclink` uses the `ld` option `--wrap` (see the ld(1) man page) to interpose monitoring code between your application and various process, thread, and signal control operations, *e.g.* `fork`, `pthread_create`, and `sigprocmask` to name a few. For some compilers, *e.g.*, IBM's XL compilers and Pathscale's compilers, interprocedural optimization interferes with the `--wrap` option and prevents `hpclink` from working properly. If this is the case, `hpclink` will emit error messages and fail. If you want to use `hpclink` with such compilers, sadly, you must turn off interprocedural optimization.

Note that interprocedural optimization may not be explicitly enabled during your compiles; it might be implicitly enabled when using a compiler optimization option such as `-fast`. In cases such as this, you can often specify `-fast` along with an option such as `-no-ipa`; this option combination will provide the benefit of all of `-fast`'s optimizations *except* interprocedural optimization.