Using HPCTOOLKIT with MPI Programs

The HPCTOOLKIT Team

March 31, 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 paper describes how to use HPCTOOLKIT with MPI programs.

1 Introduction

HPCTOOLKIT's measurement tools collect data on each process and thread of an MPI program. HPCTOOLKIT can be used with pure MPI programs as well as hybrid programs that use OpenMP or pthreads for multi-threaded parallelism.

HPCTOOLKIT supports C, C++ and Fortran MPI programs. It has been successfully tested with MPICH, MVAPICH and OpenMPI and should work with almost all MPI implementations.

2 Running and Analyzing MPI Programs

Q: How do I launch an MPI program with hpcrun?

A: For dynamically-linked binaries, use a command line similar to the following example.

```
mpirun -np num hpcrun -e EVENT:count ... program arg ...
```

Note: The MPI launch command (`mpirun`, `mpiexec`, *etc.*) goes on the outside with its options first, then `hpcrun` and its options on the inside, and finally the application program and its command-line arguments last.

Q: How do I compile and run a statically-linked MPI program?

A: On systems such as Compute-Node Linux and BG/P that run only statically-linked binaries on the compute nodes, use `hpclink` to build a statically-linked version of your application with the HPCTOOLKIT library linked in. For example,

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

Then, set the CSPROF_OPT_EVENT environment variable in the launch script before running the application.

```
export CSPROF_OPT_EVENT="PAPI_TOT_CYC@4000000"
mpiexec -n 64 myprog arg ...
```

See the *Using* HPCTOOLKIT *with Statically-Linked Programs* guide for more information.

Q: What files does `hpcrun` produce for an MPI program?

A: In this example, `s3d_f90.x` is the Fortran S3D program compiled with OpenMPI and run with the command line "`mpiexec -n 4 hpcrun -e PAPI_TOT_CYC:2500000 ./s3d_f90.x`". This produced 12 files in the following abbreviated `ls` listing.

```
krentel 1889240 Feb 18  s3d_f90.x-000000-000-72815673-21063.hpcrun
krentel    9848 Feb 18  s3d_f90.x-000000-001-72815673-21063.hpcrun
krentel 1914680 Feb 18  s3d_f90.x-000001-000-72815673-21064.hpcrun
krentel    9848 Feb 18  s3d_f90.x-000001-001-72815673-21064.hpcrun
krentel 1908030 Feb 18  s3d_f90.x-000002-000-72815673-21065.hpcrun
krentel    7974 Feb 18  s3d_f90.x-000002-001-72815673-21065.hpcrun
krentel 1912220 Feb 18  s3d_f90.x-000003-000-72815673-21066.hpcrun
krentel    9848 Feb 18  s3d_f90.x-000003-001-72815673-21066.hpcrun
krentel  147635 Feb 18  s3d_f90.x-72815673-21063.log
krentel  142777 Feb 18  s3d_f90.x-72815673-21064.log
krentel  161266 Feb 18  s3d_f90.x-72815673-21065.log
krentel  143335 Feb 18  s3d_f90.x-72815673-21066.log
```

Here, there are four processes and two threads per process. Looking at the file names, `s3d_f90.x` is the name of the program binary, `000000-000` through `000003-001` are the MPI rank and thread numbers, and `21063` through `21066` are the process IDs.

We see from the file sizes that OpenMPI is spawning one helper thread per process. Technically, the smaller `.hpcrun` files imply only a smaller calling-context tree (CCT), not necessarily fewer samples. But in this case, the helper threads are not doing much work.

Q: Do I need to include anything special in the source code?

A: Just one thing. Early in the program, preferably right after `MPI_Init()`, the program should call `MPI_Comm_rank()` with communicator `MPI_COMM_WORLD`. Nearly all MPI programs already do this, so this is rarely a problem. For example, in C, the program might begin with:

```
int main(int argc, char **argv)
```

```
{
    int size, rank;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    ...
}
```

Note: The first call to `MPI_Comm_rank()` should use `MPI_COMM_WORLD`. This sets the process's MPI rank in the eyes of `hpcrun`. Other communicators are allowed, but the first call should use `MPI_COMM_WORLD`.

Also, the call to `MPI_Comm_rank()` should be unconditional, that is all processes should make this call. Actually, the call to `MPI_Comm_size()` is not necessary (for `hpcrun`), although most MPI programs normally call both `MPI_Comm_size()` and `MPI_Comm_rank()`.

Q: What MPI implementations are supported?

A: Although the matrix of all possible MPI variants, versions, compilers, architectures and systems is very large, HPCTOOLKIT has been tested successfully with MPICH, MVAPICH and OpenMPI and should work with most MPI implementations.

Q: What languages are supported?

A: C, C++ and Fortran are supported.

3 Building and Installing HPCToolkit

Q: Do I need to compile HPCTOOLKIT with any special options for MPI support?

A: No, HPCTOOLKIT is designed to work with multiple MPI implementations at the same time. That is, you don't need to provide an `mpi.h` include path, and you don't need to compile multiple versions of HPCTOOLKIT, one for each MPI implementation.

The technically-minded reader will note that each MPI implementation uses a different value for `MPI_COMM_WORLD` and may wonder how this is possible. `hpcrun` (actually `libmonitor`) waits for the application to call `MPI_Comm_rank()` and uses the same communicator value that the application uses. This is why we need the application to call `MPI_Comm_rank()` with communicator `MPI_COMM_WORLD`.