Maskeraid v1.2

Mask Aid for Affinity Binding (Scheduling)
Document Revision 1.0
July 16, 2016

Kent Milfeld
milfeld@tacc.utexas.edu

High Performance Computing
Texas Advanced Computing Center
The University of Texas at Austin

Abstract

Maskeraid is a set of tools to help researchers discover the binding of their processes(MPI ranks)/threads(OpenMP threads) in a parallel environment. Maskeraid has the following components:

- Stand-alone executables to report default masks for execution of an OpenMP, MPI or Hybrid in an interactive or batch environment.
- API for instrumenting applications to report affinity masks from within the program.
- Utilities: timers, process/thread binder, routine to set core load.

Our intention is to create a tool that provides simple-to-understand affinity information. Bug reports and feedback on usability and improvements are welcome; send to milfeld@tacc.utexas.edu.

If you use maskeraid, cite:

github.com/TACC/maskeraid, "Maskeraid: An Aid for Reporting Affinity Masks", Texas Advanced Computing Center (TACC), Kent F. Milfeld. [1]

Contents

1 Installation

Maskeraid is easy to build. A make command will build stand-alone executables, and a library for those who want to instrument their code.

Download the github repository [1] by clicking on the "Download ZIP" file, and expand it in a convenient location.

```
unzip  maskeraid-master.zip
```

You can also clone the git repository:

```
git clone https://github.com/TACC/maskeraid
```

This will create a top level directory called `maskeraid`, with subdirectories `docs` and `src`. Change directory to the top level of maskeraid and execute make.

The executables will be placed in the `maskeraid/bin` directory and the library will be placed in `maskeraid/lib`.

[1] https://github.com/TACC/maskeraid

2 Affinity Mask from Stand-Alone Executable

The standalone executables `omp_whereami`, `mpi_whereami`, and `hybrid_whereami` are to be run in an OpenMP, MPI or hybrid environment, respectively, to obtain an output displaying the affinities for processes/threads in a batch or interactive environment.

Listing 2.1: Executables for evaluating OpenMP MPI and Hybrid masks

```
1        export OMP_NUM_THREADS=4; omp_whereami
2
3        mpirun -n 8  mpi_whereami
4
5        export OMP_NUMPTHREAD=4; mpirun -n 8  hybrid_whereami
```

In batch mode run the whereami executable before your production code, or simply run the whereami executeable in lieu of the program executable by commenting it out as shown here:

Listing 2.2: Invoking mask report inside code

```
1        #!/bin/bash
2        #SLURM  -n 16 -N 1
3            ...
4        #Batch Script for TACC machine
5           ...
6        #ibrun ./my_mpi_exec
7         ibrun ./mpi_whereami 30
```

The whereami puts a load on the ranks for 10 seconds (default), so that `top` can be run on a node to observe the core occupation. You can change the load time (seconds) with the `nsec` argument on the command line. In the last command above the MPI processes are loaded for 30 seconds. (Execute `top`, and then press the 1 key without hitting the return key, to change the display to show the percentage load on each core (or hardware thread when hyperthreading is on).

How to read the output: Output 1 is for a Stampede MPI run with 8 MPI processes.

The rows of the matrix are labeled by the ranks and the columns represent CPU_ids from 0 to ntasks-1, beginning with the first column of the matrix. The CPU_ids are grouped and labeled in units of 10s, and separated by a bar (—).

A HYPHEN at row, column position (x,y) means that rank x is not to execute on CPU_id y. A NON-HYPHEN character in the matrix means that the MPI rank (row value) is allowed to execute in the CPU_id (column value). We use only a single character to represent occupations, so that we can use this tool with many-cores systems with over 50 cores and hundreds of hardware threads.

What do the numbers in the matrix mean? To make it easy to determine the CPU_id value in the matrix, the first digit of CPU_id (core) number is printed in the matrix. For CPU_ids larger than 9, the "10s" position value can be read from the column header,the groups are separated by bars (—). For instance the second set of values 0, 1, 2, 3, 4, 5 along the diagonal of the matrix represent CPU_id's 10, 11, 12, 13, 14, and 15– the "10" in the header "— 10 —" should be added to the unit digits within the matrix to get the CPU_id value.

3 Using the Maskeraid Library

The basic operations used for reporting masking in the `whereami` programs were collected into a library, so that users could instrument there own applications to display the masks of MPI processes and OpenMP threads.

3.1 Get Masks Inside a Program

To report the masking from inside a program, include the maskeraid API routine, `omp_report_mask()`, `mpi_report_mask()`, or `hybrid_report_mask()`, within an OpenMP parallel region, after MPI has been initialized, or within an OpenMP region of a hybrid program, respectively. These functions don't require any arguments or include files:

- omp_report_mask()
- mpi_report_mask()
- hybrid_report_mask()

However, to report masks for hybrid code, it may be necessary to initialized MPI with the `MPI_Init_thread()` routine. View the `whereami` codes to see how easy it is to include them in your own program. The snippets below show how they are to be used:

Listing 3.1: Invoking mask report inside code

```
1  // Pure OpenMP code
2     #pragma omp parallel
3     {
4        omp_report_mask();
5      ...
6     }
7
8  // Pure MPI code
9     MPI_Init(&argc, &argv);
10
11       mpi_report_mask();
12    ...
13    MPI_Finalize();
14
15 // Hybrid code
16    MPI_Init_thread(&argc, &argv, MPI_THREAD_MULTIPLE, &provided);
17
18    mpi_report_mask();  // helpful: reports ONLY MPI process masks
19
20   #pragma omp parallel private(thrd,nthrds,ierr)
21   {
22      hybrid_report_mask();
23      ...
24   }
25   ...
26   MPI_Finalize();
```

3.2 Useful Utilities

- load_cpu_nsec(nsec) — Puts load on process/thread for nsec seconds
- map_to_cpuid(cpu_id) — Sets process/thread to execute on cpuid
- gtod_timer() — Easy to use Get Time of Day clock
- tsc() — Returns time stamp counter value

A thread or process that calls `load_cpu_nsec(int nsec)` will execute integer operations (a load) for nsec seconds. nsec must be zero or a positive integer. Use the `cmdln_get_nsec_or_help(int * nsec, int argc, char * argv[])` function to extract an integer from the command line for nsec (e.g. a.out 10).

A thread or process that calls `map_to_cpuid(int cpu_id)` will assign the calling thread or process to execute on CPU_id, by setting the appropriate bit in the scheduling mask. For example, in a parallel region the unique thread_id returned from `omp_get_thread_num()` can be used in an arithmetic operation (linear, modulo, etc.) to create a unique CPU_id to be executed on.

The function `gtod\_timer()` returns a double precision number with the number of wall-clock seconds since the previous call. The first call sets the time

to zero. The function uses the Unix `gettimeofday` utility, and can be called from C/C++ and Fortran. See comments in the code for more details.

The function `tsc()` returns an 8-byte integer (Unix) time stamp count from the `rdtsc` instruction. Use this to capture the difference between the counts for determining the cost a of a small set of operations (instructions, code statements). This is not meant to be used as as a general timer, since the processor frequency may change.

3.3 Other things you should know

For MPI executables, compiled with IMPI (Intel MPI), you can set I_MPI_DEBUG=4, and the execution will report the masking for list of CPU_ids for each MPI process.

Bibliography

[1] maskeraid is a set of executables and routines for reporting affinity information. https://github.com/tacc/maskeraid