epcc

OpenMP Microbenchmarks V2.0

[OpenMP Microbenchmarks V2.0] [OpenMP Microbenchmarks V1.0]

Downloading, compiling and running the benchmark codes

Downloading

Download version 2.0 of the benchmark suite from below as a gzip'd tar file:

Fortran 90 version

C version

Compiling

Fortran 90 version

1. Unpack the tar file
2. Edit the `Makefile` as follows:
   - Set `FC` to the Fortran compiler you wish to use (e.g. `f90`)
   - Set `FFLAGS` to any required Fortran compiler flags to enable processing of OpenMP directives (e.g. `-mp` or `-omp`).
     Standard optimisation is also recommended (e.g. `-O`).
   - Set `LDFLAGS` to any required Fortran linker flags
   - Set `CPP` to the local C-Preprocessor (e.g. `/usr/local/bin/cpp`) to make the Fortran compiler invoke `cpp` on `.f90` files
   - OpenMP 2.0 features can be invoked by setting the flag `OMPFLAG = -DOMPVER2`. If this flag is left unset then OpenMP 1.0 compatibility is ensured.
   - Set `CLOCKFLAG` to the clock routine you wish to use. Available options are:

`-DOMPCLOCK`	`OMP_GET_WTIME`	OpenMP clock
`-DF90CLOCK`	`system_clock`	F90 system clock

     If available, it is recommended that the OpenMP function `OMP_GET_WTIME()` should be used. Alternatively, you may wish to supply your own version of `getclock.f` to use a different clock routine. If

another routine is used, it is recommended that the clock routine returns 64-bit floating point values and is accurate to the nearest microsecond.

- Set `LIBS` to any required libraries

3. Set the value of the parameter `mhz` in `benchdata.f90` to the CPU clock rate in MHz
4. Type `make`

C version

1. Unpack the tar file
2. Edit the `Makefile` as follows:
   - Set `CC` to the C compiler you wish to use (e.g. `cc`)
   - Set `CFLAGS` to any required C compiler flags to enable processing of OpenMP directives (e.g. `-mp` or `-omp`).
     Standard optimisation is also recommended (e.g. `-O`).
   - Set `LDFLAGS` to any required C linker flags
   - Set `CPP` to the local C-Preprocessor (e.g. `/usr/local/bin/cpp`) to make the C compiler invoke `cpp` on `.c` and `.h` files
   - OpenMP 2.0 features can be invoked by setting the flag `OMPFLAG = -DOMPVER2`. If this flag is left unset then OpenMP 1.0 compatibility is ensured.
   - Set `CLOCKFLAG` to the clock routine you wish to use. Available options are:

`-DOMPCLOCK`	`OMP_GET_WTIME`	OpenMP clock
`-DGTODCLOCK`	`get_time_of_day`	Uses get-time-of-day clock routine

     If available, it is recommended that the OpenMP function `OMP_GET_WTIME()` should be used. Alternatively, you may wish to supply your own version of `getclock.f` to use a different clock routine. If another routine is used, it is recommended that the clock routine returns 64-bit floating point values and is accurate to the nearest microsecond.

   - Set `LIBS` to any required libraries.
3. Set the value of the macro `MHZ` in `schedbench.c` to the CPU clock rate in MHz
4. Type `make`

Running

1. Set the environment variable OMP_NUM_THREADS to the number of threads you wish to use.

2. Run one of the synchronisation benchmark `syncbench`, the scheduling benchmark `schedbench`, or the array benchmark for array size = n, `arraybench_n`.
3. For the array benchmark a separate executable is created for each array size specified in the `Makefile` . To investigate the variation of overhead time with array size it is necessary to run each executable `arraybench_n` separately.

Notes

1. It is common to observe significant variability between the overhead values obtained on different runs of the benchmark programs. Therefore, it is advisable to run each benchmark, say, 20 times and average the results obtained.
2. You should use whatever mechanisms are at your disposal to ensure that threads have exclusive or almost exclusive access to processors. You should rejects runs where the standard deviation or number of outliers is large: this is a good indication that the benchmark did not have almost exclusive access to processors.
3. The array benchmark has numerous executables - one for each array size specified in the `Makefile`. Depending on the memory available to you may wish alter the number of executables created, or the array sizes.
4. Warning: If you use the Intel Fortran Compiler for Linux Version 7.1 the names of the `.mod` files are capitalised automatically. This is a "feature" of the compiler and cannot be avoided. A work around is to capitalise all the `.mod` files in the `Makefile`. Version 8.0 of the Intel Fortran Compiler for Linux does not exhibit this feature.

Webpage maintained by www@epcc.ed.ac.uk