



CESGA Alliance

UPC Operations Microbenchmarking Suite 1.1 User's manual

Authors:

PhD. Guillermo López Taboada¹

Damián Álvarez Mallón²

¹taboada@udc.es
²dalvarez@cesga.es

Contents

1	Contact	2
2	Acknowledgments	2
3	Files in this benchmarking suite	2
4	Operations tested	3
5	Customizable parameters	5
5.1	Compile time	5
5.2	Run time	6
6	Compilation	8
7	Timers used	9
8	Output explanation	9

1 Contact

You can contact us at:

Galicia Supercomputing Center (CESGA)
<http://www.cesga.es>
Santiago de Compostela, Spain
upc@cesga.es

PhD. Guillermo Lopez Taboada
Computer Architecture Group (CAG)
http://gac.des.udc.es/index_en.html
University of A Coruña, Spain
taboada@udc.es

2 Acknowledgments

This work was funded by Hewlett-Packard Spain and partially supported by the Ministry of Science and Innovation of Spain under Project TIN2007-67537-C03-02 and by the Galician Government (Xunta de Galicia, Spain) under the Consolidation Program of Competitive Research Groups (Ref. 3/2006 DOGA 12/13/2006). We gratefully thank Brian Wibecan for his comments and for share with us his thoughts and knowledge. Also, we thank Jim Bovay for his support, and CESGA, for providing access to the FinisTerra supercomputer.

3 Files in this benchmarking suite

- `doc/manual.pdf`: This file. User's manual.
- `COPYING` and `COPYING.LESSER`: Files containing the use and redistribution terms (license).
- `changelog.txt`: File with changes in each release.
- `Makefile`: Makefile to build the benchmarking suite. It relies on the `src/Makefile` file.
- `src/affinity.upc`: UPC code with affinity-related tests.
- `src/config/make.def.template.*`: Makefile templates for HP UPC and Berkeley UPC.
- `src/config/parameters.h`: Header with some customizable parameters.
- `src/defines.h`: Header with needed definitions.
- `src/headers.h`: Header with HUCB functions headers.
- `src/mem_manager.upc`: Memory-related functions for allocation and freeing.

- `src/UOMS.upc`: Main file. It contains the actual benchmarking code.
- `src/init.upc`: Code to initialize some structures and variables.
- `src/Makefile`: Makefile to build the benchmarking suite.
- `src/timers/timers.c`: Timing functions.
- `src/timers/timers.h`: Timing functions headers.
- `src/utils/data_print.upc`: Functions to output the results.
- `src/utils/utilities.c`: Auxiliary functions.

4 Operations tested

- `upc_forall` (read elements of a shared array)
- `upc_forall` (write elements of a shared array)
- `upc_forall` (read+write elements of a shared array)
- `for` (read elements of a shared array)
- `for` (write elements of a shared array)
- `for` (read+write elements of a shared array)
- `upc_barrier`
- `upc_all_broadcast`
- `upc_all_scatter`
- `upc_all_gather`
- `upc_all_gather_all`
- `upc_all_permute`
- `upc_all_exchange`
- `upc_all_reduceC`
- `upc_all_prefix_reduceC`
- `upc_all_reduceUC`
- `upc_all_prefix_reduceUC`
- `upc_all_reduceS`
- `upc_all_prefix_reduceS`

- `upc_all_reduceUS`
- `upc_all_prefix_reduceUS`
- `upc_all_reduceI`
- `upc_all_prefix_reduceI`
- `upc_all_reduceUI`
- `upc_all_prefix_reduceUI`
- `upc_all_reduceL`
- `upc_all_prefix_reduceL`
- `upc_all_reduceUL`
- `upc_all_prefix_reduceUL`
- `upc_all_reduceF`
- `upc_all_prefix_reduceF`
- `upc_all_reduceD`
- `upc_all_prefix_reduceD`
- `upc_all_reduceLD`
- `upc_all_prefix_reduceLD`
- `upc_memcpy (remote)`
- `upc_memget (remote)`
- `upc_mempu (remote)`
- `upc_memcpy (local)`
- `upc_memget (local)`
- `upc_mempu (local)`
- `memcpy (local)`
- `memmove (local)`
- `upc_memcpy_async (remote)`
- `upc_memget_async (remote)`
- `upc_mempu_async (remote)`
- `upc_memcpy_async (local)`

- `upc_memget_async` (local)
- `upc_memput_async` (local)
- `upc_memcpy_asynci` (remote)
- `upc_memget_asynci` (remote)
- `upc_memput_asynci` (remote)
- `upc_memcpy_asynci` (local)
- `upc_memget_asynci` (local)
- `upc_memput_asynci` (local)
- `upc_all_alloc`
- `upc_free`

The `upc_forall` and `for` benchmarks test the performance of accesses to a shared `int` array in read, write and read+write operations. The `upc_forall` benchmark distributes the whole workload across threads, whereas in the `for` benchmark all the work is performed by thread 0. This is useful for testing the speed of remote accesses and optimization techniques such as coalescing. The operation performed in read is a sum of a variable in the stack and the current element in the array, to prevent the compiler from dropping the first $N - 1$ iterations. The operation performed in write is a simply update of the elements with its position in the array. The operation performed in read+write is a sum of the current element and its position in the array.

In bulk memory transfer operations there are two modes: remote and local. Remote mode will copy data from one thread to another, whereas local mode, will copy data from one thread to another memory region with affinity to the same thread.

5 Customizable parameters

5.1 Compile time

In the `src/config/parameters.h` file you can customize some parameters at compile time. They are:

- `NUMCORES`: If defined it will override the detection of the number of cores. If not defined the number of cores is set through the `sysconf(_SC_NPROCESSORS_ONLN)` system call.
- `ASYNC_MEM_TEST`: If defined asynchronous memory transfer tests will be built. Default is defined.
- `ASYNCI_MEM_TEST`: If defined asynchronous memory transfer with implicit handlers tests will be built. Default is defined.
- `MINSIZE`: The minimum message size to be used in the benchmarking. Default is 4 bytes.
- `MAXSIZE`: The maximum message size to be used in the benchmarking. Default is 16 megabytes.

5.2 Run time

The following flags can be used at run time in the command line:

- `-help`: Print usage information and exits.
- `-version`: Print UOMS version and exits.
- `-off_cache`: Enable cache invalidation. Be aware that the cache invalidation greatly increases the memory consumption. Also, note that for block sizes smaller than the cache line size it will not have any effect.
- `-warmup`: Enable a warmup iteration.
- `-reduce_op OP`: Choose the reduce operation to be performed by `upc_all_reduceD` and `upc_all_prefix_reduceD`. Valid operations are:
 - `UPC_ADD` (default)
 - `UPC_MULT`
 - `UPC_LOGAND`
 - `UPC_LOGOR`
 - `UPC_AND`
 - `UPC_OR`
 - `UPC_XOR`
 - `UPC_MIN`
 - `UPC_MAX`
- `-sync_mode MODE`: Choose the synchronization mode for the collective operations. Valid modes are:
 - `UPC_IN_ALLSYNC|UPC_OUT_ALLSYNC` (default)
 - `UPC_IN_ALLSYNC|UPC_OUT_MYSYNC`
 - `UPC_IN_ALLSYNC|UPC_OUT_NOSYNC`
 - `UPC_IN_MYSYNC|UPC_OUT_ALLSYNC`
 - `UPC_IN_MYSYNC|UPC_OUT_MYSYNC`
 - `UPC_IN_MYSYNC|UPC_OUT_NOSYNC`
 - `UPC_IN_NOSYNC|UPC_OUT_ALLSYNC`
 - `UPC_IN_NOSYNC|UPC_OUT_MYSYNC`
 - `UPC_IN_NOSYNC|UPC_OUT_NOSYNC`
- `-msglen FILE`: Read user defined problem sizes from `FILE` (in bytes). If specified it will override `-minsize` and `-maxsize`
- `-minsize SIZE`: Specifies the minimum block size (in bytes). Sizes will increase by a factor of 2

- `-maxsize SIZE`: Specifies the maximum block size (in bytes)
- `-time SECONDS`: Specifies the maximum run time in seconds for each block size. Disabled by default. Important: this setting will not interrupt an ongoing operation
- `-input FILE`: Read user defined list of benchmarks to run from FILE. Valid benchmark names are:

- `upc_forall_read`
- `upc_forall_write`
- `upc_forall_readwrite`
- `for_read`
- `for_write`
- `for_readwrite`
- `upc_barrier`
- `upc_all_broadcast`
- `upc_all_scatter`
- `upc_all_gather`
- `upc_all_gather_all`
- `upc_all_exchange`
- `upc_all_permute`
- `upc_memget`
- `upc_memput`
- `upc_memcpy`
- `local_upc_memget`
- `local_upc_memput`
- `local_upc_memcpy`
- `memcpy`
- `memmove`
- `upc_all_alloc`
- `upc_free`
- `upc_all_reduceC`
- `upc_all_prefix_reduceC`
- `upc_all_reduceUC`
- `upc_all_prefix_reduceUC`
- `upc_all_reduceS`
- `upc_all_prefix_reduceS`
- `upc_all_reduceUS`

- upc_all_prefix_reduceUS
- upc_all_reduceI
- upc_all_prefix_reduceI
- upc_all_reduceUI
- upc_all_prefix_reduceUI
- upc_all_reduceL
- upc_all_prefix_reduceL
- upc_all_reduceUL
- upc_all_prefix_reduceUL
- upc_all_reduceF
- upc_all_prefix_reduceF
- upc_all_reduceD
- upc_all_prefix_reduceD
- upc_all_reduceLD
- upc_all_prefix_reduceLD
- upc_memget_async
- upc_memput_async
- upc_memcpy_async
- local_upc_memget_async
- local_upc_memput_async
- local_upc_memcpy_async
- upc_memget_asynci
- upc_memput_asynci
- upc_memcpy_asynci
- local_upc_memget_asynci
- local_upc_memput_asynci
- local_upc_memcpy_asynci

6 Compilation

To compile the suite you have to setup a correct `src/config/make.def` file. Templates are provided to this purpose. The needed parameters are:

- **CC**: Defines the C compiler used to compile the C code. Please note this has nothing to do with the resulting C code generated from the UPC code if your UPC compiler is a source to source compiler.

- **CFLAGS**: Defines the C flags used to compile the C code. Please note this has nothing to do with the resulting C code generated from the UPC code if your UPC compiler is a source to source compiler
- **UPCC**: Defines the UPC compiler used to compile the suite
- **UPCFLAGS**: Defines the UPC compiler flags used to compile the suite. Please note you should not specify the number of threads flag at this point
- **UPCLINK**: Defines the UPC linker used to link the suite
- **UPCLINKFLAGS**: Defines the UPC linker flags used to link the suite
- **THREADS_SWITCH**: Defines the correct switch to set the desired number of threads. It is compiler dependant, and also includes any blank space after the switch

Once you have set up your `make.def` file you can compile the suite.

For a static thread setup type:

```
make NTHREADS=NUMBER_OF_UPC_THREADS
```

E.g., for 128 threads:

```
make NTHREADS=128
```

For a dynamic thread setup just type:

```
make
```

7 Timers used

This suite uses high-resolution timers in IA64 architecture. In particular it uses the Interval Timer Counter (`AR.ITC`). For other architectures it uses the `hpupc_ticks_now` if you are using HP UPC, or `bupc_ticks_now` if you are using Berkeley UPC, whose precision depends on the specific architecture. If none of this requirements are met the suite uses the default `gettimeofday` function. However, the granularity of this function only allows to measure microseconds, rather than nanoseconds.

8 Output explanation

This is an output example of the broadcast:

```
#-----  
# Benchmarking upc_all_broadcast  
# #processes = 2  
#-----  
#bytes #repetitions t_min[nsec] t_max[nsec] t_avg[nsec] BW_aggregated[MB/sec]  
4 20 19942 48820275 2463315.85 0.00  
8 20 19942 22922 21457.25 0.70  
16 20 19942 22397 21420.10 1.43  
32 20 19942 22235 21626.35 2.88  
64 20 20277 33610 22886.00 3.81
```

128	20	20285	22812	21676.60	11.22
256	20	20767	22845	22230.50	22.41
512	20	20767	23020	22314.85	44.48
1024	20	22777	29255	24169.85	70.01
2048	20	23705	25425	24603.85	161.10
4096	20	24562	27097	26437.60	302.32
8192	20	29885	33205	32174.35	493.42
16384	20	42492	44735	43919.35	732.49
32768	10	68317	70052	69490.00	935.53
65536	10	121610	123837	122635.00	1058.42
131072	10	227550	231515	229323.50	1132.30
262144	10	437645	444740	441354.00	1178.86
524288	10	861287	871700	867619.70	1202.91
1048576	5	1702722	1704420	1703642.40	1230.42
2097152	5	3417170	3435637	3429128.40	1220.82
4194304	5	6830267	6839535	6834224.40	1226.49
8388608	2	13434382	13469047	13451715.00	1245.61
16777216	2	27310152	27343357	27326755.00	1227.15
33554432	1	54294385	54294385	54294385.00	1236.02

The header indicates the benchmarked function and the number of processes involved. The first column shows the block size used for each particular row. The second column is the number of repetitions performed for that particular message size. The following three columns are, respectively, the minimum, maximum and average latencies. The last column shows the aggregated bandwidth calculated using the maximum latencies. Therefore, the bandwidth reported is the minimum bandwidth achieved in all the repetitions.

Moreover, when 2 threads are used, affinity tests are performed. This way you can measure the effects of data locality in NUMA systems, if the 2 threads run in the same machine. This feature may be useful even when the 2 threads run in different machines. E.g.: Machines with non-uniform access to the network interface, like quad-socket Opteron/Nehalem-based machines, or cell-based machines like HP Integrity servers. The output of this tests is preceded with something like:

```
#-----
# using #cores = 0 and 1 (Number of cores per node: 16)
# CPU Mask: 1000000000000000 (core 0), 0100000000000000 (core 1)
#-----
```

All tests after these lines are performed using core 0 (thread 0) and core 1 (thread 1) until another affinity header is showed.