Servet Benchmark Suite

Reference Manual

Authors:
Jorge González-Domínguez
Guillermo L. Taboada
Basilio B. Fraguela
María J. Martín
Juan Touriño

Institution:
Computer Architecture Group
Department of Electronics and Systems
University of A Coruña, Spain

Date:
August 29, 2013

Contents

1 Introduction

Servet[1] is a benchmarking suite to obtain the relevant hardware parameters of clusters of multicores and, hence, support the automatic optimization of parallel codes on these architectures. The system characteristics obtained by this tool are:

- Cache sizes.
- Shared cache topology.
- Overheads because of simultaneous memory accesses in shared memory.
- Different communication layers.
- Bandwidths of all the communication layers depending on the message size.
- Communication degradation due to concurrent accesses to the network from different nodes/cores.

Servet is publicly available under GPL license at: http://servet.des.udc.es

The suite is described in:

J. González-Domínguez, G. L. Taboada, B. B. Fraguela, M. J. Martín and J. Touriño. Servet: A Benchmark Suite for Autotuning on Multicore Clusters. Proc. 24th Intl. Parallel and Distributed Processing Symp. (IPDPS'10). Atlanta, GA, USA.

[1] As this suite dissects the machines to discover their characteristics, it obtains its name from Miguel Servet, a Spanish theologian, physician, cartographer and humanist who lived in the XVIth Century and performed many dissections, being the first European to describe the function of pulmonary circulation.

2 Installation

1. Untar the archive and move into the Servet directory.
2. Update the file *SMake.inc* in order to indicate the correct path to the MPI compiler.
3. Type `make all` to build Servet.
    - The sched library is compulsory. Files will try to link to functions available in *sched.h* so the user might need indicate its path in the flags in *SMake.inc*.

3 Execution of the Benchmarks

Once the suite is installed in the machine, the benchmarks can be executed using the *bin/benchmarks.sh* script. The characteristics obtained by these benchmarks will be available in the file *config/config_system.txt* in order to be read directly by the user or by the functions of the library that will be explained in Section 5.

The resources needed to obtain all the hardware information are:

- The whole system in a SMP machine.
- The same number of nodes as the number of cores per node. For instance if the nodes are octacore, eight nodes will be necessary. If there are not so many nodes available, the tool will use the whole machine.

therefore the user must be sure that these resources are completely available before starting the execution. Then, to run the benchmarks:

1. Update the file *config/params.txt* if any of the by default options must be changed. Information about the meaning of these parameters is available in this file.
2. Type `sh bin/benchmarks.sh MPI_LOADER NNODES NCORES_PER_NODE`.

Using as parameters for this script:

- `MPI_LOADER`: As the path to the binary to execute MPI programs.
- `NNODES`: 1 or 2 if the testbed is a SMP machine or a multi-core cluster, respectively.
- `NCORES_PER_NODE`: The total number of cores that communicate through shared memory.

NOTE: The user must be sure that the configuration of MPI creates the first `NCORES_PER_NODE` *MPI processes mapped in the same node.*

4 Execution of the Examples

Some example codes are included in the directory *examples* in order to check if the compilation and the execution of the benchmarks were successfully performed. Besides, they can help to the users to know how to employ the functions of the library that will be explained in Section 5. The mechanism to use these examples is:

1. Compile the codes by typing `make all` in the directory *examples*.
2. Execute the main script by typing `sh examples/exec_examples.sh`.
3. Test the autotuning example (to check the mapping policies provided Servet according to the characteristics in *config/config_system.txt*) by typing `examples/example_automapping NP NCORES`, being:
   - `NP`: Number of processes that need to be mapped in the system.
   - `NCORES`: Total number of cores in the system. In a multi-core cluster with `N` nodes and `C` cores per node where all cores can be used, the value of `NCORES` must be `N*C`.

5 Library Documentation

The functions specified in this section allow the user to easily access the information of the system that is stored in the file *config/config_system.txt*. All programs that can work with C are able to deal with this interface.

5.1 cache.h

All of these functions use one descriptor of type `cache_desc` that keeps the information about the cache topology.

5.1.1 load_cache_info

Function to load the information about the cache topology in the descriptor.

SYNTAX:

- `int load_cache_info(cache_desc *cache_info)`

PARAMETERS:

- `OUT cache_info`: Pointer to the descriptor where the information about the cache topology will be stored. It should not have been loaded before.
- returns:
  - 0 if everything is ok.
  - < 0 if an error occurs. The exact value is -j if an error is found in the jth field of `cache_desc`.

5.1.2 get_cache_nlevels

Function that provides the number of cache levels in the system.

SYNTAX:

- `int get_cache_nlevels(cache_desc *cache_info)`

PARAMETERS:

- `IN cache_info`: Pointer to the descriptor where the information about the cache topology is stored. It must have been loaded before.
- returns: Number of cache levels.

5.1.3 get_cache_size

Function that provides the size (in bytes) of a certain cache level.

SYNTAX:

- `int get_cache_size(cache_desc *cache_info, int level)`

PARAMETERS:

- `IN cache_info`: Pointer to the descriptor where the information about the cache topology is stored. It must have been loaded before.
- `IN level`: The selected cache level.
- returns:
  - The cache size (> 0) if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

5.1.4 get_shared_cache_group_size

Function that provides the number of cores that share a certain cache level.

SYNTAX:

- `int get_shared_cache_group_size(cache_desc *cache_info, int level)`

PARAMETERS:

- `IN cache_info`: Pointer to the descriptor where the information about the cache topology is stored. It must have been loaded before.
- `IN level`: The selected cache level.
- returns:
  - The number of cores (> 0) if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

5.1.5 get_shared_cache_cores

Function to obtain the set of cores that share a certain cache level with another core specified as parameter.

SYNTAX:

- `int get_shared_cache_cores(cache_desc *cache_info, int level,`
  `int core, int *shared_cores)`

PARAMETERS:

- `IN cache_info`: Pointer to the descriptor where the information about the cache topology is stored. It must have been loaded before.
- `IN level`: The selected cache level.
- `IN core`: The core that must be in the set of cores that share the cache level.
- `OUT shared_cores`: Output array where the set of cores that share the cache level with `core` are stored. It must have been allocated with enough space to store this data before calling the function.
- returns:
  - 0 if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

5.1.6 get_shared_cache_band

Function to obtain the bandwidth acessing one cache simultaneously from several cores specified as parameter.

SYNTAX:

- `double get_shared_cache_band(cache_desc *cache_info, int level, int ncores)`

PARAMETERS:

- `IN cache_info`: Pointer to the descriptor where the information about the cache topology is stored. It must have been loaded before.
- `IN level`: The selected cache level.
- `IN ncores`: The number of cores simultaneously accessing the cache.
- returns:
  - > 0 if everything is ok. It is the bandwidth.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

5.1.7 release_cache_info

Function to release all the information stored in the descriptor.

SYNTAX:

- `void release_cache_info(cache_desc *cache_info)`

PARAMETERS:

- `IN/OUT cache_info`: Pointer to the descriptor where the information about the cache topology is stored. It must have been loaded before.

5.2 mem_over.h

All of these functions use one descriptor of type `mem_over_desc` that keeps the information about the memory overheads due to concurrent shared memory accesses.

5.2.1 load_mem_over_info

Function to load the information about the shared memory overhead in the descriptor.

SYNTAX:

- `int load_mem_over_info(mem_over_desc *mem_over_info)`

PARAMETERS:

- `OUT mem_over_info`: Pointer to the descriptor where the information about the shared memory overhead will be stored. It should not have been loaded before.
- returns:
  - 0 if everything is ok.
  - < 0 if an error occurs. The exact value is -j if an error is found in the jth field of `mem_over_desc`.

5.2.2 get_mem_over_nlevels

Function that provides the number of different shared memory overhead levels in the system.

SYNTAX:

- `int get_mem_over_nlevels(mem_over_desc *mem_over_info)`

PARAMETERS:

- `IN mem_over_info`: Pointer to the descriptor where the information about the shared memory overhead is stored. It must have been loaded before.
- returns: Number of shared memory overhead levels.

5.2.3 get_mem_over_mag

Function that provides the magnitude of a certain shared memory overhead level. The magnitude is measured as the percentage of the total memory bandwidth obtained when a pair of cores is accessing memory concurrently.

SYNTAX:

- `double get_mem_over_mag(mem_over_desc *mem_over_info, int level)`

PARAMETERS:

- `IN mem_over_info`: Pointer to the descriptor where the information about the shared memory overhead is stored. It must have been loaded before.
- `IN level`: The selected shared memory overhead level.
- returns:
  - The magnitude (> 0) if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

5.2.4 get_mem_over_group_size

Function that provides the number of cores that share a certain shared memory overhead level.

SYNTAX:

- `int get_mem_over_group_size(mem_over_desc *mem_over_info, int level)`

PARAMETERS:

- `IN mem_over_info`: Pointer to the descriptor where the information about the shared memory overhead is stored. It must have been loaded before.
- `IN level`: The selected shared memory overhead level.
- returns:
  - The number of cores (> 0) if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

5.2.5 get_mem_over_cores

Function to obtain the set of cores that share a certain shared memory overhead level with another core specified as parameter.

SYNTAX:

- `int get_mem_over_cores(mem_over_desc *mem_over_info, int level,`
  `int core, int *shared_cores)`

PARAMETERS:

- `IN mem_over_info`: Pointer to the descriptor where the information about the shared memory overhead is stored. It must have been loaded before.
- `IN level`: The selected shared memory overhead level.
- `IN core`: The core that must be in the set of cores that share the memory overhead level.

- `OUT shared_cores`: Output array where the set of cores that share the memory overhead level with `core` are stored. It must have been allocated with enough space to store this data before calling the function.
- returns:
  - 0 if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

5.2.6 get_mem_over_group_mag

Function that provides the magnitude of a certain shared memory overhead level when a certain number of cores that share that level are accessing memory concurrently. The magnitude is measured as the percentage of the total memory bandwidth obtained.

SYNTAX:

- `double get_mem_over_group_mag(mem_over_desc *mem_over_info, int level, int ncores)`

PARAMETERS:

- `IN mem_over_info`: Pointer to the descriptor where the information about the shared memory overhead is stored. It must have been loaded before.
- `IN level`: The selected shared memory overhead level.
- `IN ncores`: Number of cores of the shared memory level that access memory concurrently.
- returns:
  - The magnitude (> 0) if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

5.2.7 release_mem_over_info

Function to release all the information stored in the descriptor.

SYNTAX:

- `void release_mem_over_info(mem_over_desc *mem_over_info)`

PARAMETERS:

- `IN/OUT mem_over_info`: Pointer to the descriptor where the information about the shared memory overhead is stored. It must have been loaded before.

5.3 comm.h

All of these functions use one descriptor of type `comm_desc` that keeps the information about the communications in the system.

5.3.1 load_comm_info

Function to load the information about communications in the descriptor.

SYNTAX:

- `int load_comm_info(comm_desc *comm_info)`

PARAMETERS:

- `OUT comm_info`: Pointer to the descriptor where the information about communications will be stored. It should not have been loaded before.
- returns:
  - 0 if everything is ok.
  - < 0 if an error occurs. The exact value is -j if an error is found in the jth field of `comm_desc`.

5.3.2 get_comm_intra_node_nlevels

Function that provides the number of different communication layers within one node of the system.

SYNTAX:

- `int get_comm_intra_node_nlevels(comm_desc *comm_info)`

PARAMETERS:

- `IN comm_info`: Pointer to the descriptor where the information about communications is stored. It must have been loaded before.
- returns: Number of intra-node communication layers.

5.3.3 get_comm_intra_node_lat

Function that provides the latency (in milliseconds) of a certain intra-node communication layer when sending a message with size equal to the L1 cache size.

SYNTAX:

- `double get_comm_intra_node_lat(comm_desc *comm_info, int level)`

PARAMETERS:

- `IN comm_info`: Pointer to the descriptor where the information about communications is stored. It must have been loaded before.
- `IN level`: The selected intra-node communication layer.
- returns:
  - The latency (> 0) if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

5.3.4 get_comm_inter_node_lat

Function that provides the latency (in milliseconds) of inter-node communications when sending a message with size equal to the L1 cache size.

SYNTAX:

- `double get_comm_inter_node_lat(comm_desc *comm_info)`

PARAMETERS:

- `IN comm_info`: Pointer to the descriptor where the information about communications is stored. It must have been loaded before.
- returns:
  - The latency (> 0) if everything is ok.
  - -100 if there is only one node in the system and thus there are no inter-node communications.
  - Another < 0 value if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

5.3.5 get_comm_intra_node_group_size

Function that provides the number of cores that share a certain intra-node communication layer.

SYNTAX:

- `int get_comm_intra_node_group_size(comm_desc *comm_info, int level)`

PARAMETERS:

- `IN comm_info`: Pointer to the descriptor where the information about communications is stored. It must have been loaded before.
- `IN level`: The selected intra-node communication layer.
- returns:
  - The number of cores (> 0) if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

5.3.6 get_comm_intra_node_cores

Function to obtain the set of cores that share a certain intra-node communication layer with another core specified as parameter.

SYNTAX:

- `int get_comm_intra_node_cores(comm_desc *comm_info, int level, int core, int *shared_cores)`

PARAMETERS:

- `IN comm_info`: Pointer to the descriptor where the information about communications is stored. It must have been loaded before.
- `IN level`: The selected intra-node communication layer.
- `IN core`: The core that must be in the set of cores that share the intra-node communication layer.
- `OUT shared_cores`: Output array where the set of cores that share the intra-node communication layer with `core` are stored. It must have been allocated with enough space to store this data before calling the function.
- returns:
  - 0 if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

5.3.7 get_comm_min_msg_size

Function that provides the minimum message size (in bytes) used to study the communication bandwidths in all layers.

SYNTAX:

- `int get_comm_min_msg_size(comm_desc *comm_info)`

PARAMETERS:

- `IN comm_info`: Pointer to the descriptor where the information about communications is stored. It must have been loaded before.
- returns: The minimum message size.

5.3.8 get_comm_max_msg_size

Function that provides the maximum message size (in bytes) used to study the communication bandwidths in all layers.

SYNTAX:

- `int get_comm_max_msg_size(comm_desc *comm_info)`

PARAMETERS:

- `IN comm_info`: Pointer to the descriptor where the information about communications is stored. It must have been loaded before.
- returns: The maximum message size.

5.3.9 get_comm_intra_node_band

Function that provides the bandwidth (in MB/s) in a certain intra-node communication layer when sending a message with a certain size.

SYNTAX:

- `double get_comm_intra_node_band(comm_desc *comm_info, int level, int msg_size)`

PARAMETERS:

- `IN comm_info`: Pointer to the descriptor where the information about communications is stored. It must have been loaded before.
- `IN level`: The selected intra-node communication layer.
- `IN msg_size`: The selected message size.
- returns:
  - The bandwidth (> 0) if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

5.3.10 get_comm_inter_node_band

Function that provides the bandwidth (in MB/s) when sending an inter-node message with a certain size.

SYNTAX:

- `double get_comm_inter_node_band(comm_desc *comm_info, int msg_size)`

PARAMETERS:

- `IN comm_info`: Pointer to the descriptor where the information about communications is stored. It must have been loaded before.
- `IN msg_size`: The selected message size.
- returns:
  - The bandwidth (> 0) if everything is ok.
  - -100 if there is only one node in the system and thus there are no inter-node communications.
  - Another < 0 value if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

5.3.11 release_comm_info

Function to release all the information stored in the descriptor.

SYNTAX:

- `void release_comm_info(comm_desc *comm_info)`

PARAMETERS:

- `IN/OUT comm_info`: Pointer to the descriptor where the information about the communications is stored. It must have been loaded before.

5.4 degr.h

All of these functions use one descriptor of type `degr_desc` that keeps the information about the communication degradation in the system.

5.4.1 load_degr_info

Function to load the information about communications in the descriptor.

SYNTAX:

- `int load_degr_info(degr_desc *degr_info)`

PARAMETERS:

- `OUT degr_info`: Pointer to the descriptor where the information about communication degradation will be stored. It should not have been loaded before.
- returns:
  - 0 if everything is ok.
  - < 0 if an error occurs. The exact value is -j if an error is found in the jth field of `degr_desc`.

5.4.2 get_max_access_network_degr

Function that provides the maximum number of cores that has been studied for the access to network degradation.

SYNTAX:

- `int get_max_access_network_degr(degr_desc *degr_info)`

PARAMETERS:

- `IN degr_info`: Pointer to the descriptor where the information about communications is stored. It must have been loaded before.
- returns: The maximum number of cores.

5.4.3 get_network_access_ratio

Function that provides the ratio between the highest communication bandwidth and the real bandwidth when certain number of cores per node are used.

SYNTAX:

- `int get_network_access_ratio(degr_desc *degr_info, int ncores_per_node)`

PARAMETERS:

- `IN degr_info`: Pointer to the descriptor where the information about communication degradation is stored. It must have been loaded before.
- `IN ncores_per_node`: The selected number of cores per node.
- returns:
  - The ratio (> 0) if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

5.4.4 get_max_dist_network_degr

Function that provides the maximum distance that has been studied for the network degradation.

SYNTAX:

- `int get_max_dist_network_degr(degr_desc *degr_info)`

PARAMETERS:

- `IN degr_info`: Pointer to the descriptor where the information about communications is stored. It must have been loaded before.
- returns: The maximum distance.

5.4.5 get_network_dist_ratio

Function that provides the ratio between the highest communication bandwidth and the real bandwidth when concurrent messages are sent to nodes with certain distance.

SYNTAX:

- `int get_network_dist_ratio(degr_desc *degr_info, int dist)`

PARAMETERS:

- `IN degr_info`: Pointer to the descriptor where the information about communication degradation is stored. It must have been loaded before.
- `IN dist`: The selected distance between the communicated nodes.
- returns:
  - The ratio (> 0) if everything is ok.
  - < 0 if a parameter error occurs. The exact value is -j if the wrong parameter is the jth one.

5.4.6 release_degr_info

Function to release all the information stored in the descriptor.

SYNTAX:

- `void release_degr_info(degr_desc *degr_info)`

PARAMETERS:

- `IN/OUT degr_info`: Pointer to the descriptor where the information about the communications is stored. It must have been loaded before.

5.5 automapping.h

This file offers a function to obtain correct mapping policies according to the information stored in *config/config_system.txt*. Two possible mapping policies are available:

- `SERVET_COMM_PRIOR`: It is focused on minimizing the impact of sharing caches and memory overheads and then, when possible, it tries to improve the latencies and bandwidths of the communications.
- `SERVET_MEM_PRIOR`: It is focused on minimizing the comunication costs and then, when possible, it tries to reduce the impact of concurrent shared memory accesses.

5.5.1 get_mapping_policy

Function that obtains the appropriate process mapping according to the hardware parameters detected by the benchmarks.

SYNTAX:

- `void get_mapping_policy(int np, int ncores, SERVET_PRIOR prior, int *policy)`

PARAMETERS:

- `IN np`: Number of processes that need to be mapped in the system.
- `IN ncores`: Total number of available cores. In a multicore cluster with n nodes and c cores per node where all cores are available for execution, the value of `ncores` must be $n * c$.
- `IN prior`: Identifier of the type of mapping policy that must be applied (`SERVET_MEM_PRIOR` or `SERVET_COMM_PRIOR`).
- `OUT policy`: Output array where the process mapping is stored. Entry i indicates the number of core where process i should be mapped. It must have been allocated with `np` elements before calling the function.