Exploring Memory Allocators in the context of Non-uniform memory access systems

VINEET PATEL[1]

[1] University of Illinois at Chicago

Compiled November 26, 2020

In general, in a computer system with non-uniform memory access (NUMA), it is desirable for a task to have memory access on the local node, in order to minimze memory latency. Throughout this semester, profiling tools were used to determine that, traditional memory allocators, like glibc's malloc implementation, do use local node allocation.

https://github.com/Vineet77/CS-398-numa

INTRODUCTION

In a NUMA (non-uniform memory access), a processor core may either access its own local memory (memory that is on its own socket), or it may access remote memory (memory that is on another socket). Since local memory is closer to the processor unit, the memory latency is smaller when accessing local memory, compared to accessing remote memory. In an ideal situation, whenever memory is allocated, it should be done as close as possible to the processing unit as possible (ideally on the local node itself)

1. MACHINE SPECIFICATION

All programs were executed in the context of a Linux mutli-core machine, part of UIC Bits lab. The machine had the following technical specificiations:

- Kernel: Linux 4.15.0-65-generic
- Distro: Ubuntu 18.04.3 LTS
- Total RAM: 96 GB
- Number of Numa nodes: 4
- CPU model: Intel(R) Xeon(R) CPU E5-4620 0 @ 2.20GHz (Jaketown)
- Number of CPUs: 64

PROFILING

In order to measure access to local and remote memory access, the perf stat command is used to collect performance counters. It is important to remember that each CPU have their own unique set of perf events. In order to firgure out which events are relevant to for a specific model, consute the Intel guide on performance events. The following is relevant for Jaketown Intel Xeon model.

In order to measure local DRAM accesses, the following events were measured:

- offcore_response.all_demand_mlc_pref_reads.llc_miss.local_dram
- offcore_response.demand_code_rd.llc_miss.local_dram
- offcore_response.demand_data_rd.llc_miss.local_dram
- offcore_response.pf_l2_data_rd.llc_miss.local_dram

The 4 measurements together measure memory response (in response to demands for code, data, prefetch requests) that miss the last layer cache (llc) and come from the local DRAM.

To measure remote DRAM accesses, the following events were measured:

- offcore_response.demand_code_rd.llc_miss.remote_dram
- offcore_response.demand_data_rd.llc_miss.remote_dram
- offcore_response.pf_l2_data_rd.llc_miss.remote_dram

These 3 events help measure remote DRAM responses to any data request that miss last layer cache (llc).

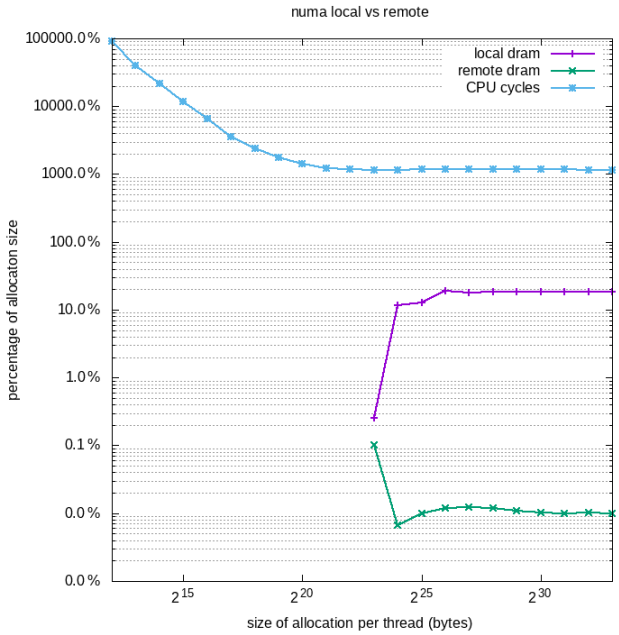
EXPERIMENT

The goals of these experiments was to see the nature of memory allocation for different memory allocators. The two primary memory allocators that were used are glibc's malloc, and lib numa allocator. The lib numa was used to explicitly control which node memory would be allocated on and which node a particular thread would run on.

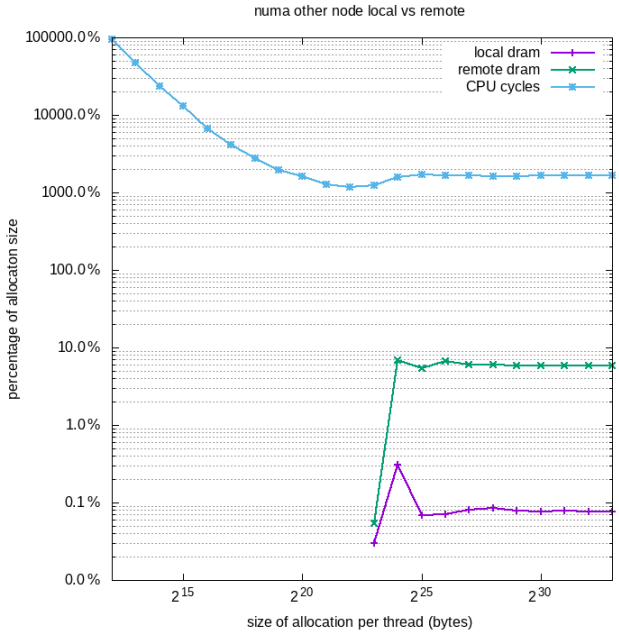
In each experiment, a set size of memory is allocated. Then in order to have both memory reads and writes, every value of each bytes is incremented by one. bytes are incremented in succession, so there is definitely a caching effect. The caching comes from memory locality, as when a single byte is requested, the cache line brings in adjacent bytes in the address space.

DATA

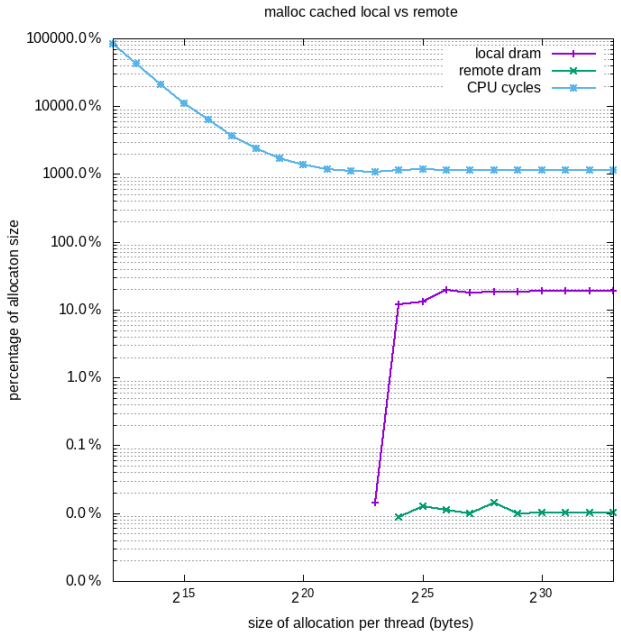
The first experiment, each node has a thread assigned to it. Using the lib numa library, memory is specifically allocated locally. Due to caching effect, local DRAM accesses only average around 20% of the total memory allocated and show up once the size exceeded 8MB.



Next, the program explicitly allocated memory to the node where the thread was not running. It is clear that remote DRAM memory accesses were performed.



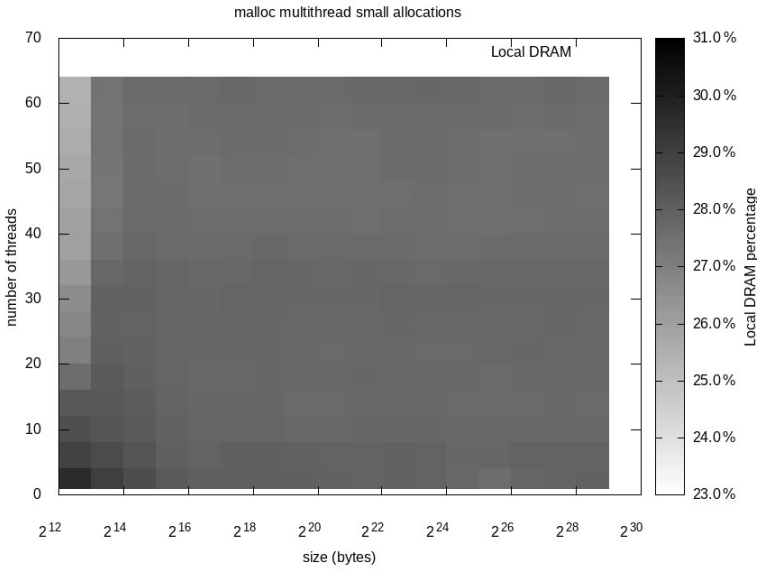
It seems that malloc allocates memory close to explicitly allocating on the local node.

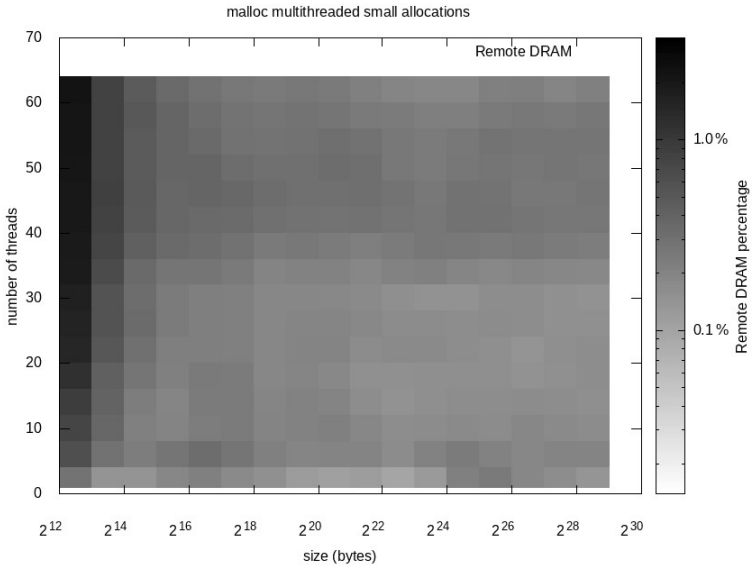


MUTLI-THREADED DATA

In set set of experiments, each thread (the number of threads varies) makes a number of small allocations of a specific allocations. The total size of memory each thread allocates total 500 MB. As before, every byte value was incremented 10 times, The purpose of this setup was to see if the memory allocation behavior changed if smaller allocations were made in the context of a mutli-threaded application.

For the glibc malloc allocator, even with small allocations (from a page size to 500 MB), malloc still seems to rely on local memory. Remote memory access is a small percetnage of total memory allocated, where as local memory access is consistently around 25% of total memory allocated.





CONCLUSION

It seems even in a non-uniform memory access system, memory allocators like glibc's malloc implementation, still optimally use local memory allocation. Further digging indicates that this comes from the fact that whenever the Linux kernel must allocate physical memory, it has the default policy of allocating pages on the node in which the task is currently running (for further info look into mbind and this article), It seems that if one is running a numa system, they can confortably use the glibc malloc implementation without worrying too much about performance.