numalloc: Node-Aware and Adaptive NUMA Memory Allocator

#147, 12 pages

OOPSLA Deadline: 04-16-2018, OSDI:05-03/2018

1. Introduction

How to design the new allocator?

We will try to achieve the following target.

(1) The performance will be as efficient as possible. (2) It is still For information-computable, we will achieve the following targets: We should still use the address to infer the following information, such as the size of the object and the shadow memory placement.

We don't support many bags for the same size class, just one big bag for each size class of each heap. Why it is necessary?

We would like to reduce the memory blowup as much as possible, where the memory will be returned back to the current thread's heap.

In order to support that, maybe we could have a big heap for the same size class, but different threads will start from different placement.

Is it good for the NUMA support in the future? For NUMA support, it is great if we can always return the memory back to its original

Virtual address will be divided into multiple nodes.

Then inside each step, we will divide a sub-heap into multiple mini-heaps, where each mini-heap will support one thread.

Virtual address will be divided into multiple nodes.

Then inside each per-node heap, we will further divide it into multiple mini-heaps, where each mini-heap will support one thread in order to reduce the possible contention.

For each per-thread heap, we will have two free-lists for each size class. The first one will be used for the current thread, without the use of lock at all. The second one will be utilized for returning memory from other threads.

RTDSCP is a way to know where a thread is executed on.

[[Should we put the metadata into the corresponding node? We will minimize the lock uses for each node, since that will invoke unnecessary remote accesses as well.]]

1.1 Novelty

- We will design a node-aware memory allocator that could actually identify the memory placement by using the virtual address.
- We will minimize the synchronization, since that will impose unnecessary remote accesses.
- We will minimize remote accesses by putting the metadata into multiple nodes.
- We may support multiple types of allocation, such as block-wise false sharing memory accesses, or node-balanced memory accesses, relying on the indication from user space.
- We will balance the memory consumption over all nodes, in order to avoid the contention of memory controller [4]. At least, we could balance the memory consumption among all allocators. Ideally, it is better to balance the memory accesses.
- We may track the relationship of all objects. Then those objects will be allocated correspondingly. For instance, if the objects are allocated in the main thread, we assume that they will be accessed as a shared mode. Then we would like to allocate in the node-balanced areana initially.

2. Evaluation

2.1 Performance Evaluation

2.1.1 NUMA Architecture

We will compare with popular allocator, such as glibc, or tcmalloc, or NUMA-aware tcmalloc as well.

2.1.2 UMA Architecture

We will compare the performance even on non-numa architecture, which could be more general.

3. Related Work

General Purpose Allocators:

NUMA-aware Allocators: **[[All of the following papers are not read carefully. Most sentences are copied from the abstract]]** [6]:

[7] [1]

[3] : Observation: it should take both data locality and cache contention into account to achieve good performance, and memory management cannot be decoupled from process scheduling. We describe two scheduling algorithms: maximum-local, which optimizes for maximum data locality, and its extension, N-MASS, which reduces data locality to avoid the performance degradation caused by cache contention. N-MASS is fine-tuned to support memory management on NUMA-multicores and improves performance up to 32%, and 7% on average, over the default setup in current Linux implementations.

Other NUMA-related Systems: Memory system performance in a numa multicore multiprocessor

[5] proposes TBB-NUMA, a system that collaborates the resource management, task scheduling, and high-level parallel algorithm templates together to achieve better performance.

[2] shows that a set of simple algorithmic changes coupled with commonly available OS functionality suffice to eliminate data sharing and to regularize the memory access patterns for a subset of the PARSEC parallel benchmarks. These simple source-level changes result in performance improvements of up to 3.1X, but more importantly, they lead to a fairer and more accurate performance evaluation on NUMA-multicore systems. In the default configuration, OSs (such as Linux) tend to change the thread-to-core mapping during the execution of programs, which result in large performance variations.

References

[1] Patryk Kaminski. "numa aware heap memory manager". `http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/NUMA_aware_heap_memory_manager_article_final.pdf`, 2012.

[2] Z. Majo and T. R. Gross. (mis)understanding the numa memory system performance of multithreaded workloads. In *2013 IEEE International Symposium on Workload Characterization (IISWC)*, pages 11–22, Sept 2013.

[3] Zoltan Majo and Thomas R. Gross. Memory management in numa multicore systems: Trapped between cache contention and interconnect overhead. In *Proceedings of the International Symposium on Memory Management*, ISMM '11, pages 11–20, New York, NY, USA, 2011. ACM.

[4] Zoltan Majo and Thomas R. Gross. Memory system performance in a numa multicore multiprocessor. In *Proceedings of the 4th Annual International Conference on Systems and Storage*, SYSTOR '11, pages 12:1–12:10, New York, NY, USA, 2011. ACM.

[5] Zoltan Majo and Thomas R. Gross. A library for portable and composable data locality optimizations for numa systems. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP 2015, pages 227–238, New York, NY, USA, 2015. ACM.

[6] Takeshi Ogasawara. Numa-aware memory manager with dominant-thread-based copying gc. In *Proceedings of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09, pages 377–390, New York, NY, USA, 2009. ACM.

[7] M. M. Tikir and J. K. Hollingsworth. Numa-aware java heaps for server applications. In *19th IEEE International Parallel and Distributed Processing Symposium*, pages 108b–108b, April 2005.