Mmalloc: A Dynamic Memory Management on Many-core Coprocessor for the Acceleration of Storage-intensive Bioinformatics Application

Zihao Wang *‡, Mingzhe Zhang†, Jingrong Zhang*‡, Rui Yan *§,
Xiaohua Wan*, Zhiyong Liu*, Fa Zhang*, Xuefeng Cui¶
*High Performance Computer Research Center, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China.
†High-Throughput Computer Research Center, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China.
‡University of Chinese Academy of Sciences, Beijing, China.
§College of Computer Science and Technology, Anhui University, Hefei, China
¶Institute for Interdisciplinary Information Sciences (IIIS) Tsinghua University
wangzihao@ict.ac.cn, zhangfa@ict.ac.cn, xfcui@mail.tsinghua.edu.cn

***Abstract*—In the past decades, many applications in bioinformatics have achieved great success by extracting useful information from huge amounts of data. However, when some storage-intensive applications like BWA-MEM ported to coprocessors to accelerate, they often have memory bottleneck that severely limits program performance and scalability. While dynamic memory allocation is one of the important topics in CPU and GPU, there has been relatively little work on many-core coprocessors.**

This paper introduces Mmalloc, a fast and highly scalable allocator that accelerates storage-intensive application on many-core coprocessor. Mmalloc is the first allocator to consider the different architecture between MIC and CPU. Mmalloc removes the global heap to reduce the long-distance on-chip coherent and communication. Mmalloc uses a binary sort interval tree to manage the memory. We also separate the header information from the data area using the logical structure to keep the locality of processed data. Our results on BWA-MEM benchmarks demonstrate that Mmalloc has a better speedup and scalability comparing with the state-of-the-art allocator for CPU like Hoard on the many-core coprocessor.

***Index Terms*—memory management,many integrated core coprocessor, storage-intensive bioinformatics application, bwa-mem**

I. Introduction

Many applications in bioinformatics including computational biology of molecular structure, computational systems biology and next-generation sequencing have achieved great success by extracting useful information from vast amounts of data such as RELION [1], BLAST [2] and BWA-MEM [3]. However, the huge computational demand becomes a bottleneck for these applications. As is known to all, the general solution to reduce the associated runtimes of these applications is implementing these algorithms on parallel computer architectures such as RELION-2 [4], GPU-BLAST [5], and BWA-MEM on GPU [6].

Mingzhe Zhang and Zihao Wang have equal contribution. This work was performed while Mingzhe Zhang was a Ph.D Candidate at High Performance Computer Research Center, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China.

Different applications have distinct characteristics with data-intensive or storage-intensive. If BWA-MEM was ported to MIC or GPU directly, many dynamic requests of global memory would reduce the performance. Although some existing methods like Hoard [7] or SFMalloc [8] which designed a scalable memory allocator have already existed, these methods only considered a small number of threads (no more than 20 processors or 50 threads) either in Hoard or SFMalloc. MIC applies a similar instruction set with CPU, but it contains up to 61 cores, and the thread number is about 244 (61 cores with each core having four hardware threads).

In this paper, we present a new dynamic memory allocator on Xeon Phi card named Mmalloc, which is first designed for many-core coprocessor considering the characteristics of the architecture. We use a new logical structure to manage the memory which is friendly to the storage-intensive application because it can handle the memory increment easier than others. We separate the header information from the data area to keep the locality of management information and data which is vital on coprocessor. At the same time, the use of logical structures can search and delete data in high efficiency. We also port a widely used storage-intensive application in bioinformatics called BWA-MEM as a benchmark to compare the performance of Mmalloc with the series malloc and Hoard which is a multithread allocator for CPU.

II. Motivation

The BWA-MEM is the latest recommend method for high-quality aligning sequence reads against a large reference genome such as human [3]. A general procedure of this algorithm is to align millions of DNA sequence reads against a reference genome obtained by next-generation sequencing (NGS). The BWA-MEM algorithm can be divided into three steps and shown in Fig. 1.

In traditional methods, the serial memory allocators will use a doubly linked lists to manage the heap such as allocating a

978-1-5386-5488-0/18/$31.00 ©2018 IEEE

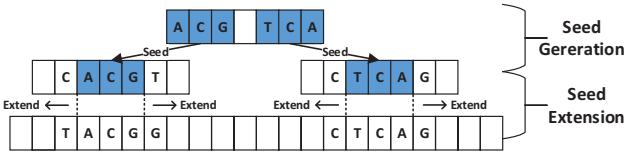


Fig. 1. BWA-MEM based on the Seed-and-Extend algorithm.

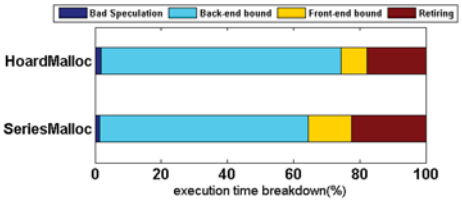


Fig. 2. Top-Down analysis breakdown for Hoard

new data block, deleting an unused block and combining some fragments. As the multithreaded programs are becoming more and more common, serial memory allocators will influence the performance seriously. Many memory allocators designed for multithreaded applications appeared for example Hoard [7], Sfmalloc [8], and Tcmalloc [9]. We analyze these various kinds of methods for dynamic memory allocations and select Hoard ported to Xeon Phi. It's much faster than built-in system allocators and more widely used by some commercial companies comparing with other published methods [10].

We ported the Hoard to Xeon Phi card to solve the memory bottleneck and to parallelize the BWA-MEM algorithm. However, there still are some problems for Hoard ported to Xeon Phi. We use Top-Down breakdown methods [11] to analyze the performance by Intel VTune Amplifier. The details of benchmark and method are described in Section V-C. In Fig. 2, although Hoardmalloc has the high percentage of *Retiring*, the percentage of *Back-End Bound* also is very high so it indicates that the bottleneck of memory accesses still exists.

After analyzing the Hoard and experiment result, we can find some shortcomings for storage-intensive application on many-core coprocessor. It uses local heaps to avoid heap contention, and multiple threads requests the same heap simultaneously. The frequent *malloc* and *free* operators occuring in BWA-MEM make the threads frequently occupy the global heap to allocate and free superblocks located in the local heap. As the microarchitecture between CPU and MIC is different, using the global heap to distribute and reduce superblocks on Xeon Phi card will cause the redundant communication between different cores, and it will severely raise the bottleneck of memory. Secondly, the linear list data structure finds a proper block to allocate in linear time and the thread has a long list to search so the process doesn't have high efficiency. Thirdly, the block array structure is not friendly to memory increment which often used in dynamic memory allocator. Furthermore, the data structure divides the contiguous memory area into the data area and list header area. This layout destroys the locality of data on Xeon Phi card. All these concerns

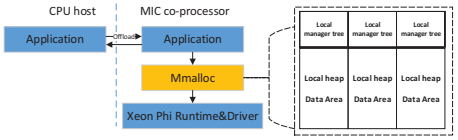


Fig. 3. Overview of Mmalloc allocator

motivated us to explore a new dynamic memory allocation algorithm suitable for many-core coprocessor to parallelize the storage-intensive application like BWA-MEM.

III. Method

To deal with the existing problems, we proposed a new dynamic memory allocator on MIC called Mmalloc. In this section, we will present the design and implementation of Mmalloc in detail to explain how to improve the parallel efficiency of storage-intensive application like BWA-MEM.

A. Overview

Fig. 3 shows an overview of the Mmalloc. We implement the method based on Xeon Phi runtime and driver functions. In the offload mode, the application can use Mmalloc by calling the *mmalloc()* and *mfree()* function. Mmalloc mainly divides the whole managed memory area into two parts: data area and management area, And the whole data area is equally divided into the C parts (the C is a constant parameter, indicating the number of local heaps). The management area is also divided equally into the same C parts. Some threads were distributed on the neighboring processors share the same local heap.

B. Storage Structures

Mmalloc classifies the whole heap memory into three categories: heapmanager, datamanager and datablock. Heapmanager and datamanager are located in the management area. Mmalloc creates the fixed number of heapmanager equal to the number of local heaps. Heapmanager is used to record the status of a local heap and organize the datamanager, so it includes the statistical information of the managed local heap and the status of datamanager. There are two parts in each datamanager. One part is used to link with other datamanager, and the other parts record the number and address of managed datablock. Datablock is the last category in Mmalloc, and they are all in the same size. The size is a power of 2 to align and balance the real requirement of a program with internal fragmentation. Without the redundant head information in datablock, it can make full use of the memory and keep the locality of data.

C. Logical Structures

Fig. 4 shows the detail of the logical structures of Mmalloc. Each heapmanager of a local heap maintains an approximately balanced binary sort interval tree to manage the heap memory. When a request for new memory comes, the heapmanager will generate a new leaf node with the index of the requested size of data. Each leaf node is a datamanager which manages a sub-area of memory.

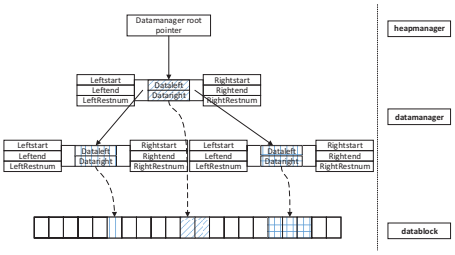


Fig. 4. Logical structure of Mmalloc allocator

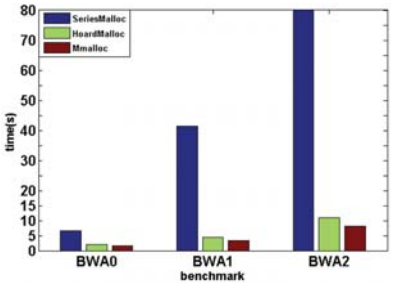


Fig. 5. Benchmark speedup

D. Mmalloc Workflow

At first, the size of memory requested decides the number of needed datablocks while the number of thread decide the local heap number using a simple hash function. If the heap lock is not occupied, the thread will get the lock and the root node from the heapmanager of the specific local heap. Heapmanager checks the status of the node is a NULL pointer or not. If the node is a NULL pointer, heapmanager inserts a free datamanager in this node and select the needed datablock number from the middle of the available datablock range. Then, the manager will return the pointer of the first datablock as a result. If the node isn't a NULL pointer, heapmanager selects one side of the binary tree which has more free space to check by comparing the sum of rest space in left and right sides. The process running recursively until the checked node is a NULL pointer.

E. Free Workflow

Because the data area does not have any redundant information, we only can make full use of the address information in the data pointer. According to the design of the storage structures, we can use the high address to get the heap number and use the low address to get the location of data area. After getting these information, specific heapmanager can find the datamanager which allocates and manages the target free datablocks by searching the binary sort tree recursively.

IV. Experimental Setup

We use a Dell Poweredge R720XD system equipped with a Intel Xeon CPU E5-2620 v2, 32 GB of RAM and a Intel Xeon Phi 5110p (KNC) MIC coprocessor, 8 GB of RAM. The coprocessor has 60 cores, each of which has 4 Hyper-Threading and includes a vector processing unit with the width of 512 bits [12].

To evaluate the usefulness of Mmalloc with respect to the motivation from Section II. We download the BWA code from [13] and analyse the BWA-MEM source code. We change the parallel programming model from Pthread to Openmp is widely used on MIC [14]. Then the BWA-MEM algorithm is ported to MIC using offload mode named BWA-MEM-MIC. The test data is downloaded from GATK website [15] named $tutorial\_6484\_FastqToSam.tar.gz$ including two sequence of files that can be aligned. We use text file named $6484\_snippet\_1.fastq$ as reference and align the part of $6484\_snippet\_2.fastq$ data against the reference. According to the different line selected from align data, we have three benchmarks named BWA0, BWA1 and BWA2.

V. Result

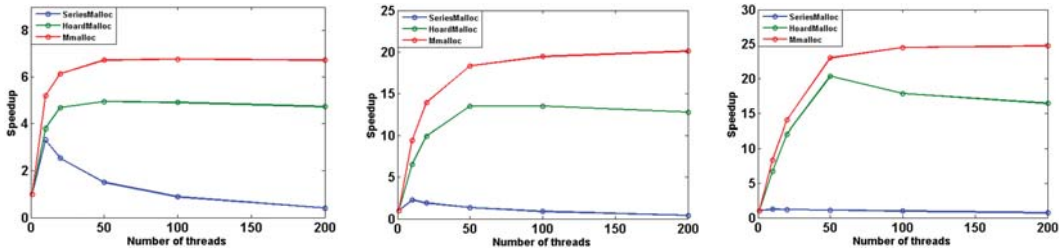
A. Speedup

We use the data of BWA0, BWA1 and BWA2 to run the BWA-MEM-MIC application to test the performance of SeriesMalloc, HoardMalloc and Mmalloc. Fig. 5 shows the specific time spent by three methods. In these test cases, we use 50 threads for three methods and the same local heap number equal to 50 for HoardMalloc and Mmalloc. The size of datablock is all 4096B. We perform the tests for five times and calculate the mean value. Moreover, to display the HoardMalloc and Mmalloc clearly, the result of SeriesMalloc in BWA2 is reduced to 80s, but the real time is 876s. In Fig. 5, Mmalloc approximately has 1.2-1.4 times of acceleration comparing with the HoardMalloc.

B. Scalability

These benchmarks also are used to measure the scalability. We use 50 local heaps. And the size of datablock is 4096B. The result of BWA0 is shown in Fig. 6(a). At first, all methods improve the performance along with the increasing number of threads. However, when the number of threads go up to 20, the performance of SeriesMalloc is influenced by the frequent memory operations. Because of the BWA0 is a small dataset, the HoardMalloc and Mmalloc all have similar scalability as the number of threads increased.

Fig. 6(b) shows the result of the BWA1 benchmark. When the thread number is smaller than 100, similar behavior between HoardMalloc and Mmalloc appeared. But it is a dataset of medium size, so we can observe that when the thread number exceeds the number of private heap number, the performance of HoardMalloc is affected by the global communication on the Xeon Phi. In Fig. 6(c) of BWA2, the scalability even drops because of the frequency behavior of *malloc()* and *free()* cause the global heap occupied frequently. Comparing with these situations, Mmalloc has better scalability and speedup than HoardMalloc in the three benchmarks of different size.



(a) BWA0 benchmark scalability (b) BWA1 benchmark scalability (c) BWA2 benchmark scalability

Fig. 6. The scalability of different benchmark

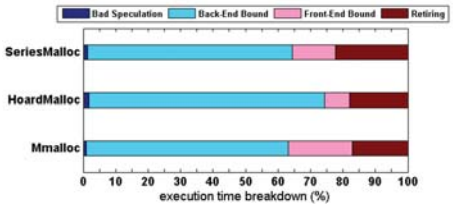


Fig. 7. Xeon Phi execution time breakdown

C. Threading breakdown

Fig. 7 shows the KNC execution time breakdown of all benchmarks [11]. We use the BWA1 benchmark, and the running configuration of memory allocator is the same with Section V-A. In Fig. 7, Mmalloc has the lowest percentage to wait for back-end stalls especially comparing with the HoardMalloc and also has the lowest bad speculation time in all three methods. The low back-end stalls indicate that Mmalloc does not aggravate the bottlenecks of storage-intensive when parallelizing the application which has many requests of memory.

VI. Conclusion

In this paper, we introduced the Mmalloc memory allocator. It is designed for accelerating storage-intensive bioinformatics application on many-core coprocessor. The different architecture between CPU and MIC makes the state-of-the-art allocator for CPU like Hoard unsuitable for MIC. We reduce the long-distance on-chip coherence and communication by removing the global heap in Hoard and use a binary sort interval tree to manage the memory. It can handle the memory increment easier and operate data in high efficiency. We separate the header information from the data area using the logical structure. We also port a widely used storage-intensive application BWA-MEM as a benchmark. Our experimental results on three different size benchmarks show that Mmalloc has better performance and scalability compared with Hoard on the many-core coprocessor. Our method is suitable for accelerating storage-intensive application on many-core architecture coprocessor.

Acknowledgment

This research is supported by the Strategic Priority Research Program of the Chinese Academy of SciencesGrant No. XDA19020400, the National Key Research and Development Program of China (No. 2017YFE0103900 and 2017YFA0504702), the NSFC projects Grant (No. U1611263, U1611261, 61472397, 61502455 and 61672493) and Special Program for Applied Research on Super Computation of the NSFC-Guangdong Joint Fund (the second phase). Some computations were performed on Tianhe-2 supercomputer at the National Supercomputer Center in Guangzhou (NSCC-GZ), China.

References

[1] S. H. Scheres, "Relion: implementation of a bayesian approach to cryo-em structure determination," *Journal of structural biology*, vol. 180, no. 3, pp. 519–530, 2012.
[2] J. Hetherington and P. Smith, *Blast and ballistic loading of structures*. CRC Press, 2014.
[3] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with bwa-mem," *arXiv preprint arXiv:1303.3997*, 2013.
[4] D. Kimanius, B. O. Forsberg, S. H. Scheres, and E. Lindahl, "Accelerated cryo-em structure determination with parallelisation using gpus in relion-2," *Elife*, vol. 5, 2016.
[5] P. D. Vouzis and N. V. Sahinidis, "Gpu-blast: using graphics processors to accelerate protein sequence alignment," *Bioinformatics*, vol. 27, no. 2, pp. 182–188, 2010.
[6] E. J. Houtgast, V. Sima, K. Bertels, and Z. AlArs, "An efficient gpuaccelerated implementation of genomic short read mapping with bwamem," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 4, pp. 38–43, 2017.
[7] E. D. Berger, K. S. McKinley, R. D. Blumofe, and P. R. Wilson, "Hoard: A scalable memory allocator for multithreaded applications," in *ACM SIGARCH Computer Architecture News*, vol. 28, no. 5. ACM, 2000, pp. 117–128.
[8] S. Seo, J. Kim, and J. Lee, "Sfmalloc: A lock-free and mostly synchronization-free dynamic memory allocator for manycores," in *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*. IEEE, 2011, pp. 253–263.
[9] S. Ghemawat and P. Menage, "Tcmalloc: Thread-caching malloc," 2009.
[10] (2017) Hoard : the memory allocator. [Online]. Available: http://hoard.org/
[11] A. Yasin, "A top-down method for performance analysis and counters architecture," in *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 35–44.
[12] S. Peng, X. Zhang, Y. Lu, X. Liao, K. Lu, C. Yang, J. Liu, W. Zhu, and D. Wei, "mamber: A cpu/mic collaborated parallel framework for amber on tianhe-2 supercomputer," in *Bioinformatics and Biomedicine (BIBM), 2016 IEEE International Conference on*. IEEE, 2016, pp. 651–657.
[13] (2018) Burrows-wheeler aligner. [Online]. Available: http://bio-bwa.sourceforge.net/
[14] J. Reinders, "An overview of programming for intel xeon processors and intel xeon phi coprocessors," *Intel Corporation, Santa Clara*, 2012.
[15] (2017) Generate an unmapped bam from fastq or aligned bam. [Online]. Available: https://software.broadinstitute.org/gatk/documentation/article?id=6484