# Multicore Processors: Architecture & Programming Project

**Project Title:** Memory Allocator for OpenMP Programs

**Project Group Number:** 2

**Project Group Members:**

1. Darshan Dinesh Kumar (dd3888, N10942768)

NYU

# Problem Definition

- Moore's Law no longer holds true
- Need for innovative solutions to augment performance
- Multicore Processors are ubiquitous, ideal candidates for performance improvements
- However, parallel applications can be inhibited by the memory allocator
- An unscalable memory allocator can be a significant bottleneck to performance
- Further, it can introduce issues such as False Sharing and Fragmentation
- Consequently, there's a need for a scalable memory allocator that can solve these issues and positively contribute to the performance of parallel applications
- This project introduces such a scalable memory allocator for parallel applications (OpenMP)

# Literature Survey

### Serial Single Heap

- Global Heap protected by a single lock
- Serializes memory operations
- Affects scalability and actively induces false sharing
- Ex: Solaris, Windows NT/2000 allocators

### Concurrent Single Heap

- Heap is a concurrent data structure (Ex: B-tree)
- Actively induces false sharing
- Expensive locks and atomic update operations

### Pure Private Heaps

- A Heap per processor
- Better suited for scalability
- Freed Memory is placed on freeing thread's heap, passively inducing false sharing
- Ex: STL's pthread_alloc, Cilk allocators

### Private Heaps with Thresholds

- A Heap per processor with limited free memory
- When free memory on a per processor heap exceeds threshold, it is moved to a shared heap, passively inducing false sharing
- Ex: Hoard, DYNIX kernel allocator

### Private Heaps with Ownership

- A Heap per processor
- Freed memory is returned to owner processor's heap, reducing false sharing
- Ex: MT-malloc, Ptmalloc, Lkmalloc which can still exhibit false sharing under certain scenarios
- Allocator implemented in this project falls under this category of per thread heaps with memory ownership which attempts to eliminate false sharing and minimize fragmentation

# Experimental Setup

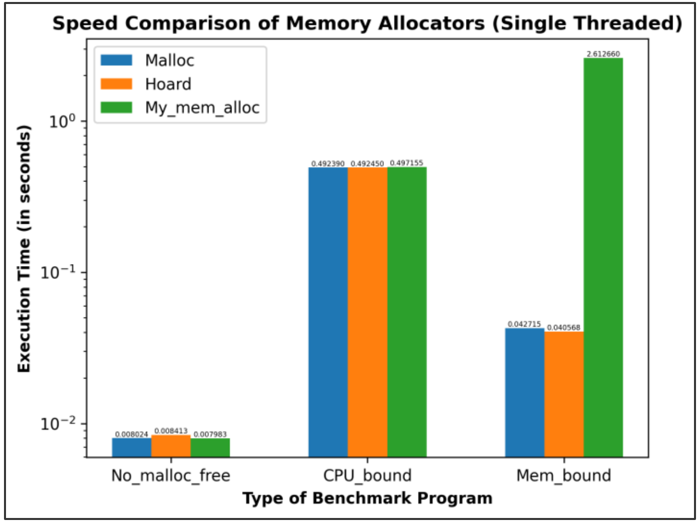
| | |
|---|---|
| Architecture | x86_64 |
| Number of CPUs | 64 |
| Number of Sockets | 4 |
| Number of Cores per Socket | 8 |
| Number of Threads per Core | 2 |
| L1d Cache | 1 MiB (64 instances) |
| L1i Cache | 2 MiB (32 instances) |
| L2 Cache | 64 MiB (32 instances) |
| L3 Cache | 48 MiB (8 instances) |
| Page Size | 4K bytes |
| Cache Alignment | 64 bytes |
| L1d Cache Line Size | 64 bytes |

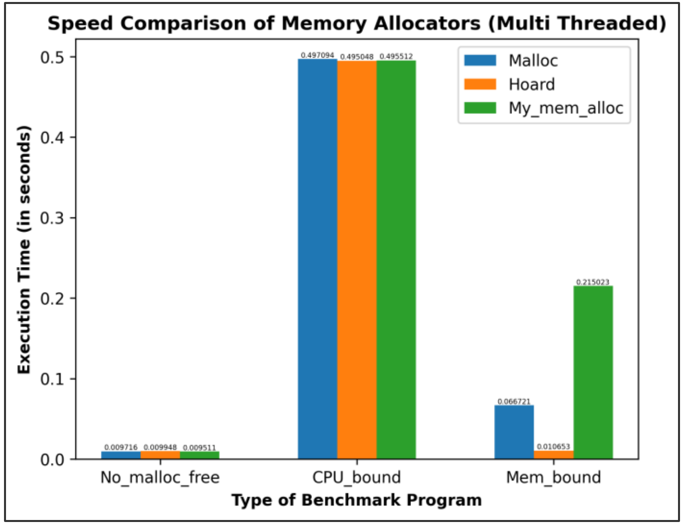
**Configuration of crunchy2 CIMS machine used for Experiments**

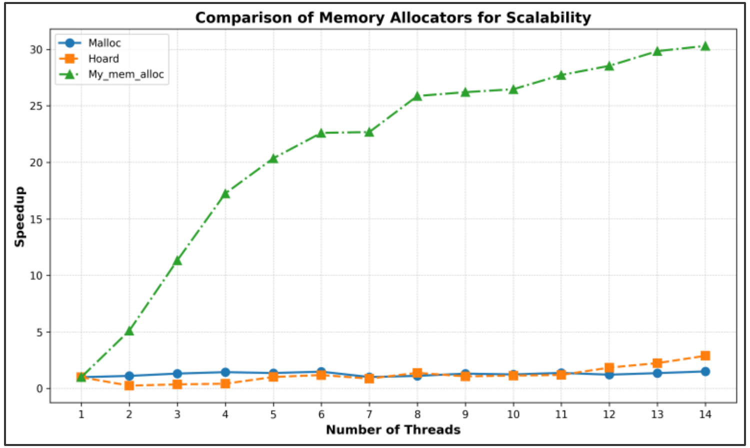
**OpenMP Benchmarks developed for Experimentation**

1. Speed
   a) No Malloc or Free
   b) CPU bound
   c) Memory bound
2. Scalability
3. False Sharing Avoidance
   a) Active False Sharing
   b) Passive False Sharing
4. Fragmentation

# Results and Analysis



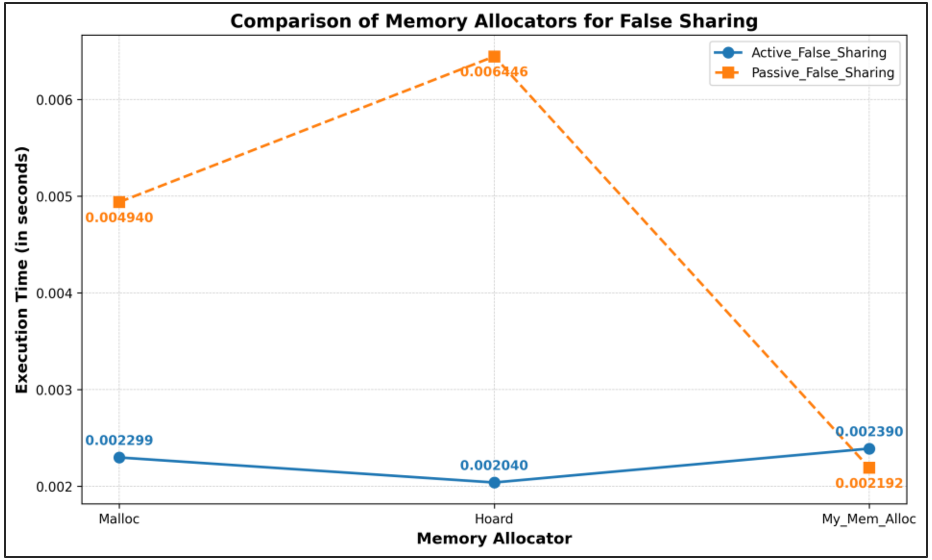



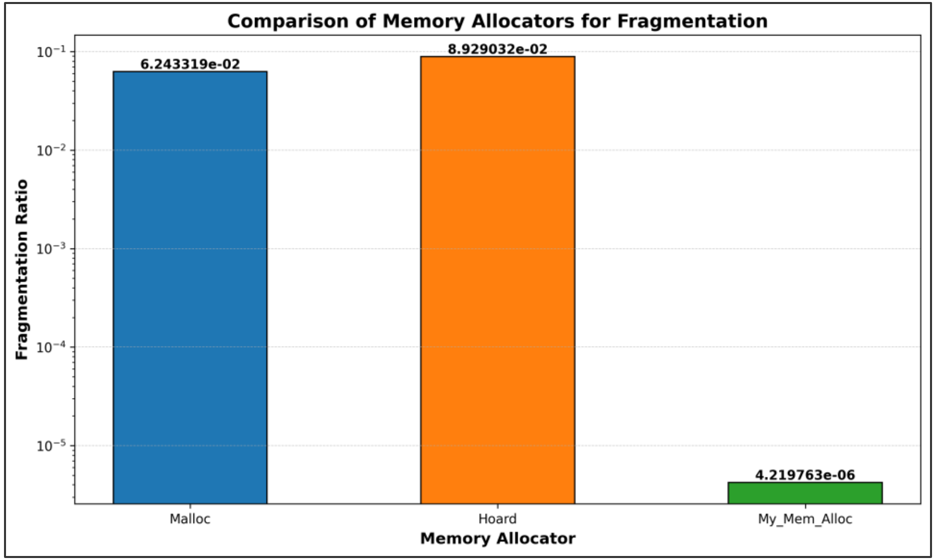




** Speedup for n threads = $\frac{\text{Execution Time for 1 thread}}{\text{Execution Time for n threads}}$

NYU

# Results and Analysis




$$\text{** Fragmentation Ratio} = \frac{\text{Max amount of memory allocated from OS}}{\text{Max amount of memory required by application}}$$

NYU

# Conclusion

1. The project introduces a scalable memory allocator using per thread heaps with memory ownership with the following objectives:
   a) Complement and Improve the performance in terms of speed and scalability of parallel applications (OpenMP)
   b) Prevent issues inherent to memory allocation like false sharing and fragmentation

2. The results generated for the developed OpenMP benchmarks are promising, especially for Scalability, False Sharing Avoidance and Low Fragmentation as compared to the Malloc and Hoard memory allocators

3. Admittedly, there is scope for Future work, especially in regards to improving the speed of the developed allocator which could involve further experimentation as follows:
   a) Alternatives to best fit algorithm for finding free blocks like first fit, worst fit, next fit etc.
   b) Unmapping based on memory usage statistics

NYU