Towards Practical Page Placement for a Green Memory Manager

17 December, 2015

Ashish Panwar, K. Gopinath

Indian Institute of Science

Bangalore, India

Agenda

- Motivation
- Background
- Challenges
- Design and Implementation
- Results
- Conclusion and Future Work

Motivation

- Memory consumes significant power in some situations (sometimes upto 40%).
- Proportion of memory power consumption increases in idle states.
- Software hardware cooperative power management (CPU and other subsystems) makes it even more critical.
- However, such cooperation for memory power optimization is still missing in traditional systems despite continuous hardware support.

Hardware Support

Partial Array Self-Refresh:

- Memory banks can be turned off to save power.
- Requires operating system support to avoid data loss.

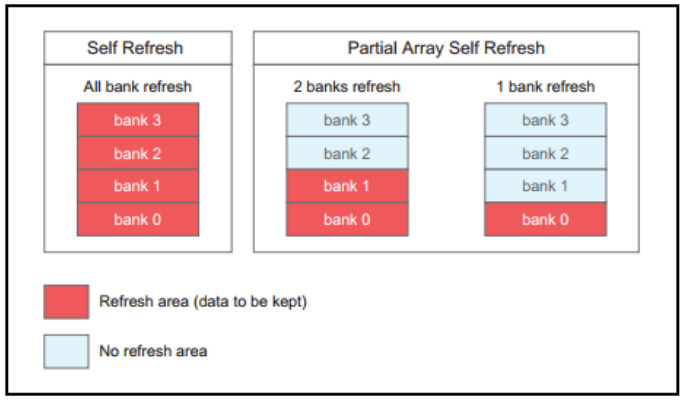


Fig 1. Overview of Partial Array Self Refresh [1]

Power State	Relative Power Consumption
Read/Write	100 %
Active-Idle	63 %
Self-Refresh	7 %
Deep-Power-Down	< 1%

Table 1. Relative Power Consumption of different power states [1]

- Power state transition of memory banks is typically transparent to software.

[1] https://www.micron.com/~/media/documents/products/technical-note/dram/e0597e10.pdf

Linux Page Allocator

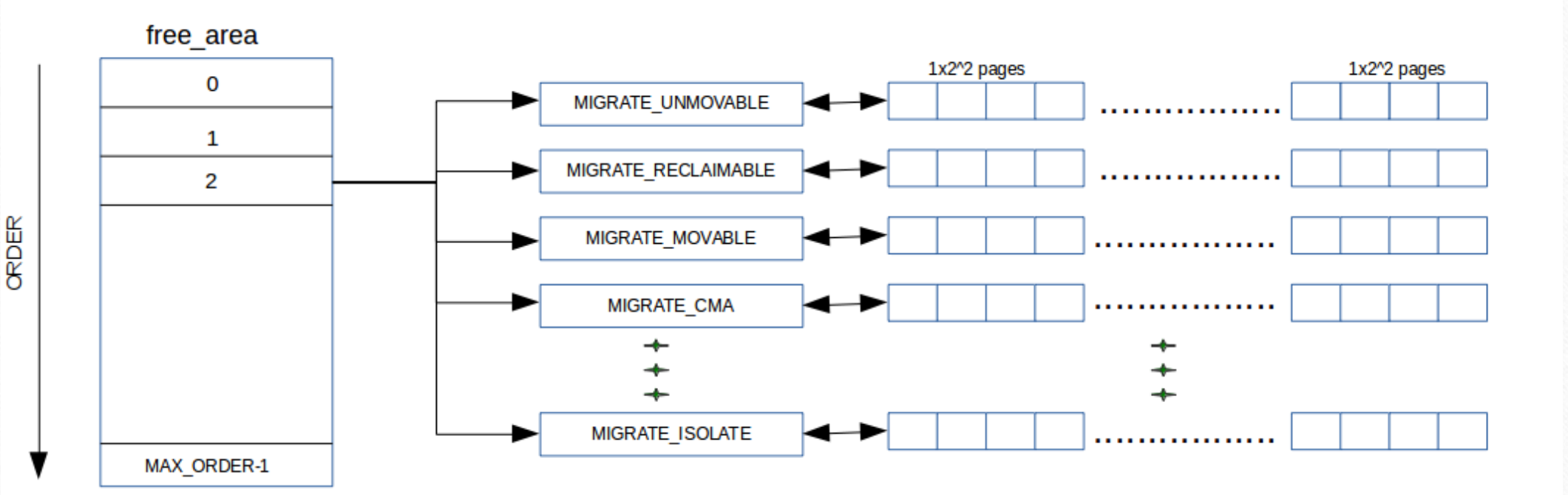


Fig 2. Layout of Linux Binary Buddy Allocator

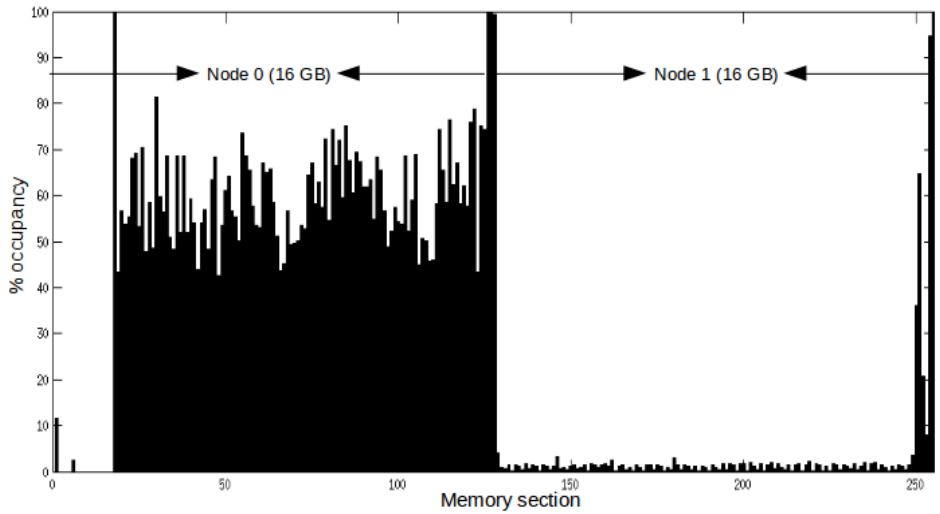
- Memory is divided in multiple zones.
- Each memory zone gets its own buddy allocator.

Role of a Memory Manager

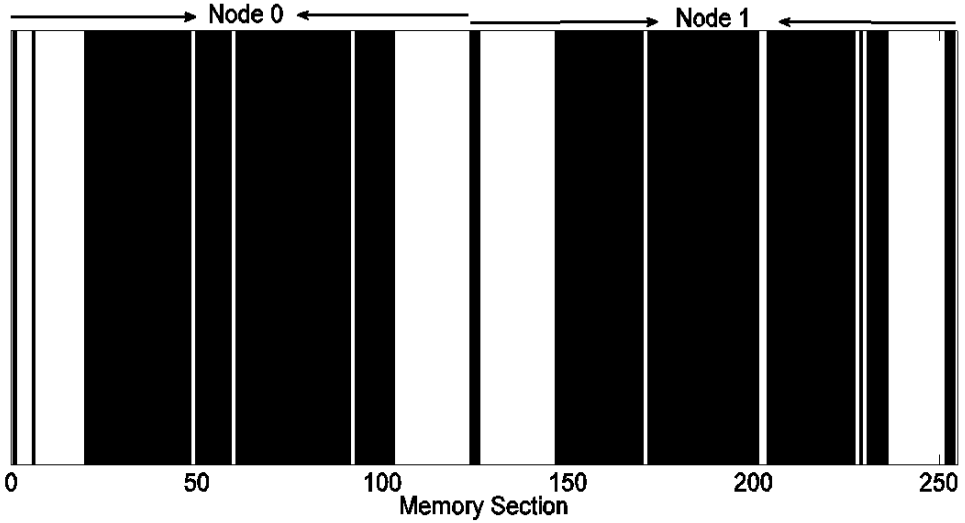
- Restrict page allocations to a subset of memory banks.
- Perform migration (or compaction) in order to limit the number of in-use memory banks in long run (should be kept to minimum as migration itself will burn some power).

Linux page allocator performs poorly for both requiremets !

Behavior of Linux Page Allocator



(a). All memory sections are in-use for even < 10 % memory usage on Node 1.



(b). Sections colored black (172/256) can not be freed because of the presence of kernel (unmovable) pages.

Fig 3. Impact of arbitrary page allocation of Linux kernel

Arbitrary page allocation causes -

- Memory references to spread over almost all memory banks (even during low workloads).
- A large proportion of memory banks to become unmovable due to the presence of kernel pages.

Challenge

- Working sets are complex, large and rapidly changing on modern hardware.
- Actual memory references are typically transparent to operating systems.

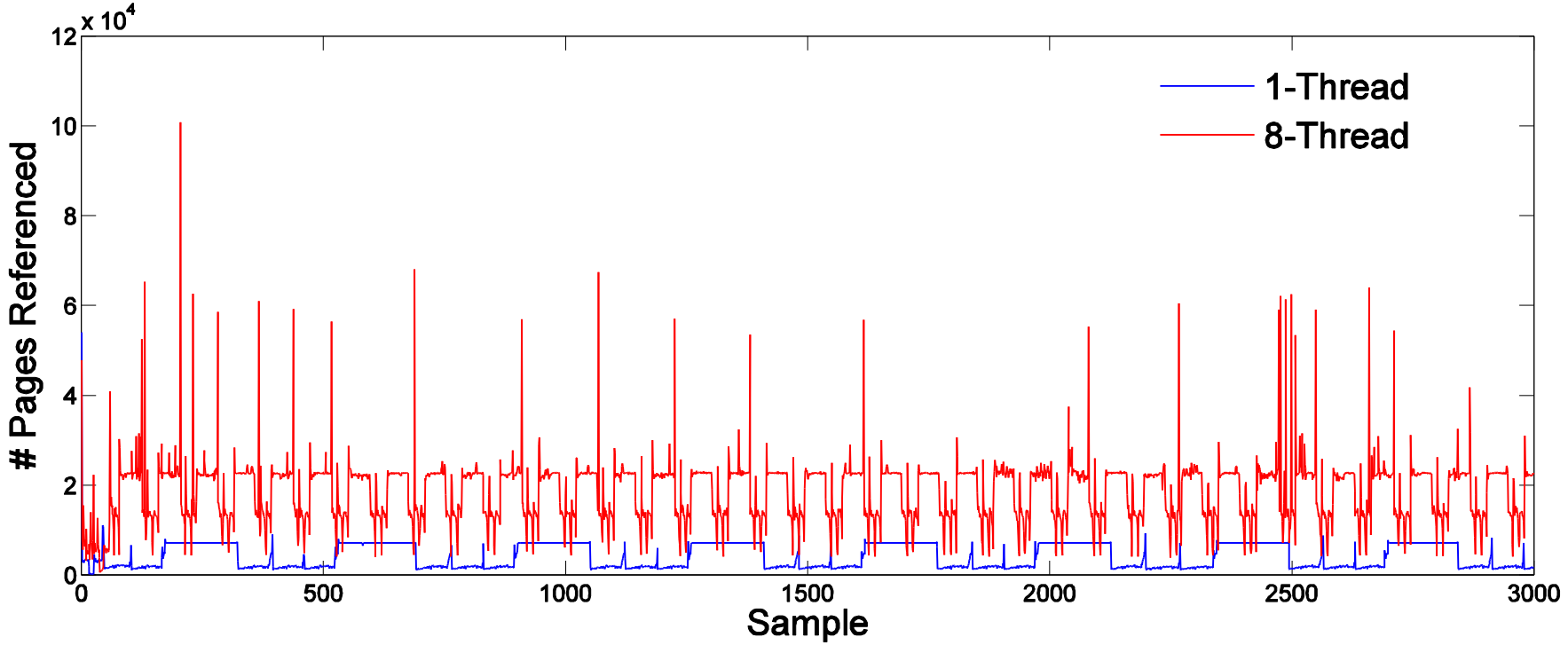


Fig 4. Working set (10ms sample) of *facesim* with different thread count.

Challenge

- Idle periods are hard to achieve as the number of cores grow large.
- Processing such information in a running system is processor (and hence power) hungry.

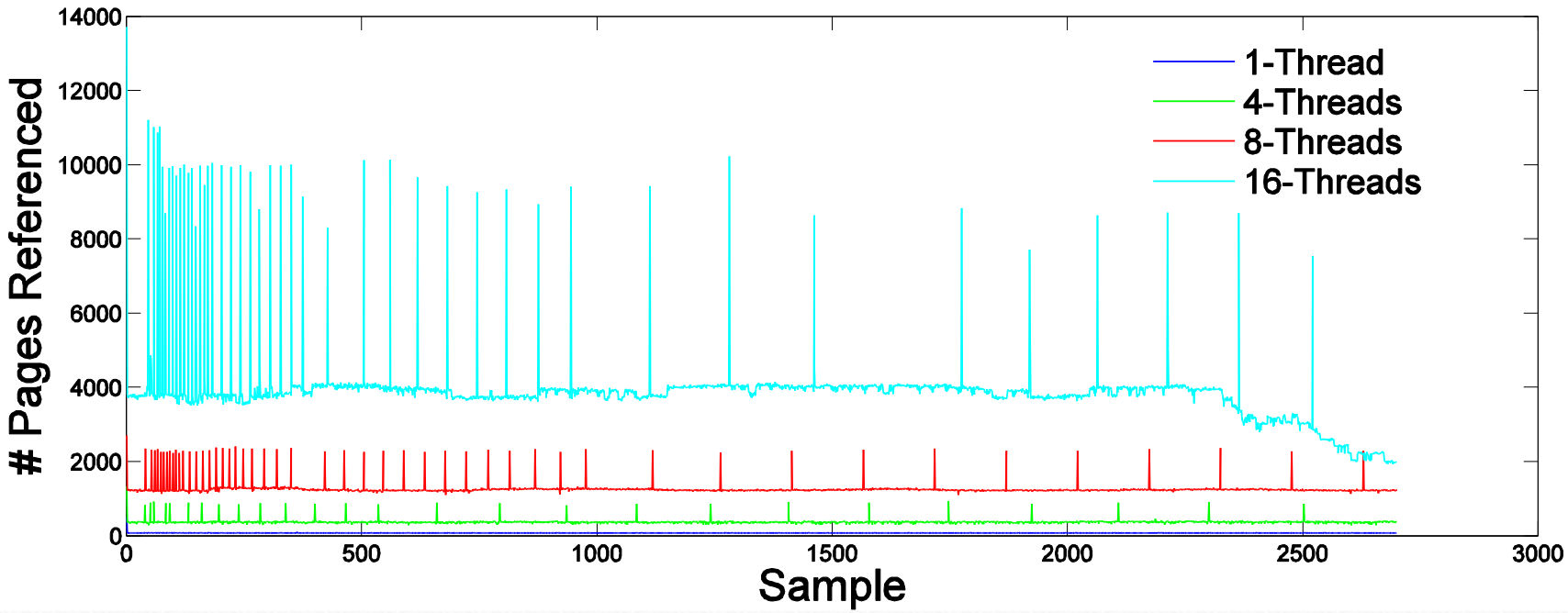


Fig 5. Working set (10ms sample) of *swaptions* with different thread count.

Array based Bank-Buddy Allocator

- Manages free pages of each memory bank individually.
- Allocation from one end of the array.
- Different directions for kernel and user memory allocations.
- Migrate when there is an opportunity.

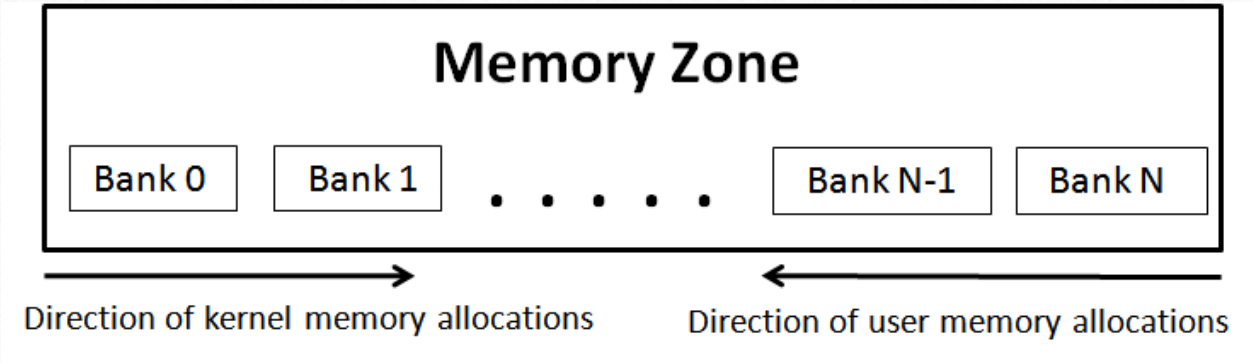
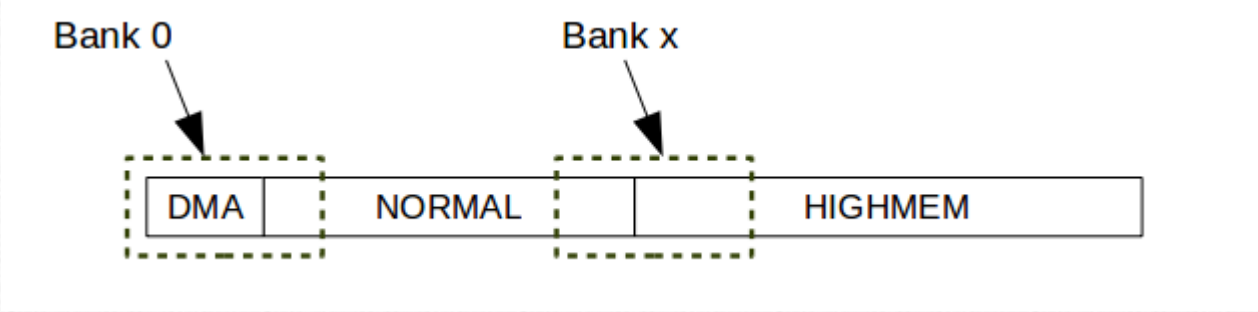
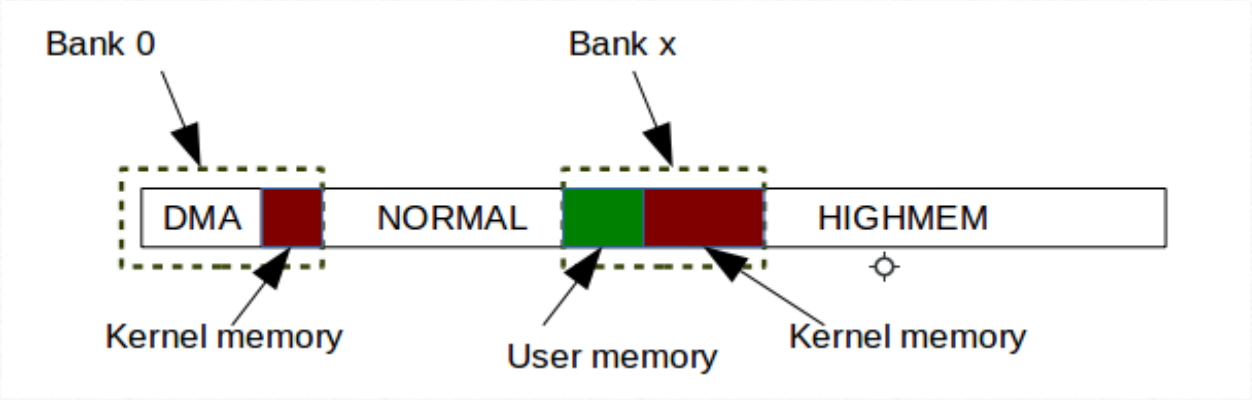


Fig 6. Array based Bank-Buddy Allocator

When a bank crosses zone boundary

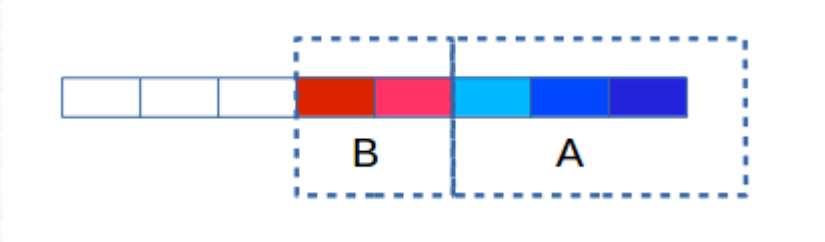


- There are at-most two such possibilities in a system.
- Always keep memory banks that cross zone boundary in-use.



Analyzing Array based Bank-Buddy Allocator

- Cost of Page Allocation : O(n), where n is the number of memory banks spanned by the zone.
- Unnecessary migration overhead.



If A exits first -

List based Bank-Buddy Allocator

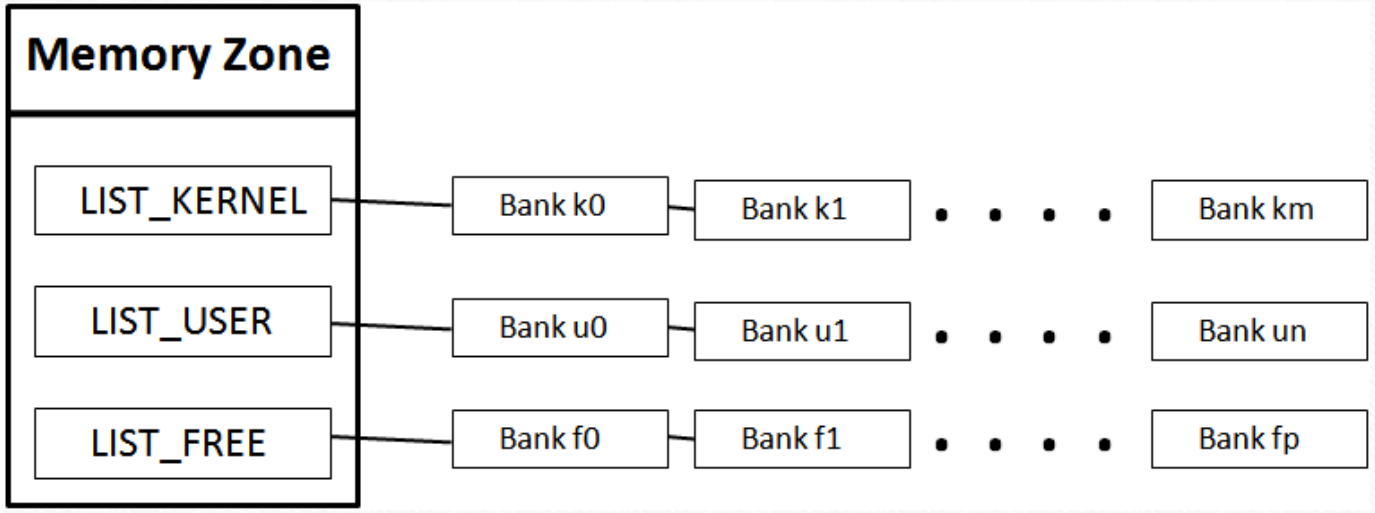


Fig 7. Layout of memory banks in List based Bank-Buddy Allocator

- Helps in reducing migration overhead but the cost of memory allocation is still high i.e., O(n) in the worst case.

Issues with per Bank Buddy Allocators

- 104 bytes per order free lists on x86_64.
- 1144 bytes for *free_area* per memory zone.
- Impact: Size of memory management structures (and hence the kernel image) grows significantly.
- MICRO-OPTIMIZATION MACRO SIDE-EFFECTS (mainly due to inefficient cache behavior).

Adaptive-Buddy Allocator

- Merge all banks belonging to same list (from list based design) together to form memory pools.
- Provides better runtime performance compared to earlier designs.
- Capacity on demand can be met by adding (or removing) banks to (or from) allocation pools.

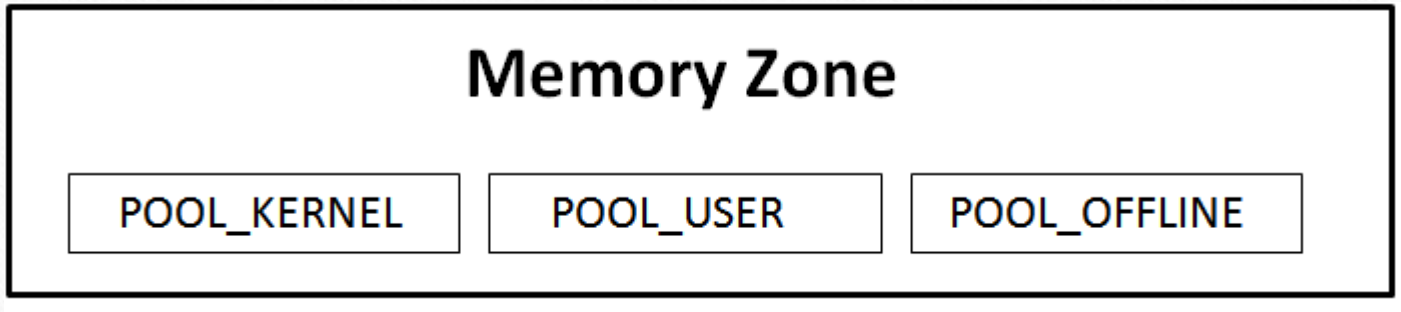
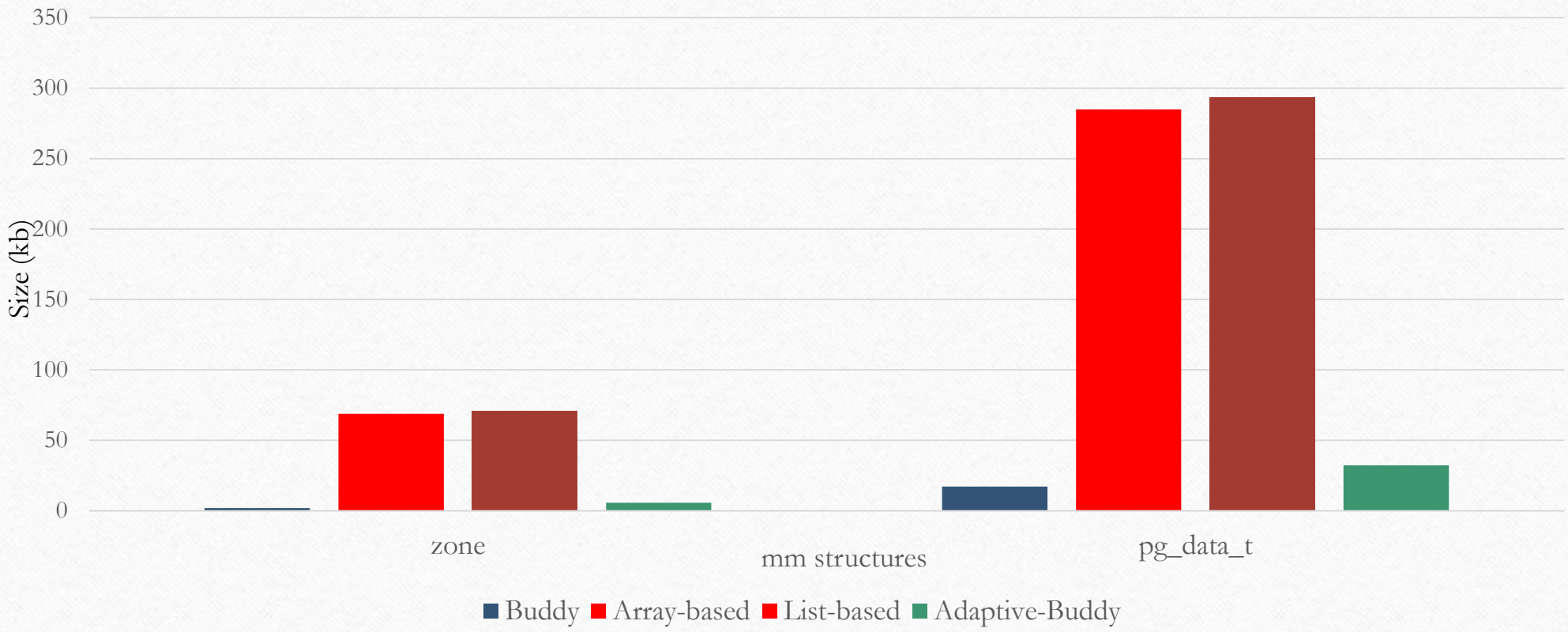


Fig 8. Layout of memory pools in Adaptive-Buddy

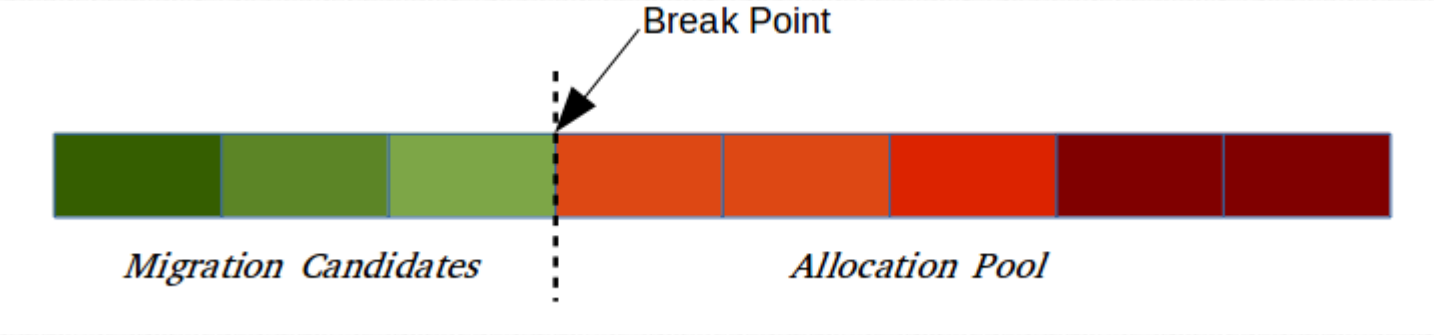
Page Allocation : O(1)
Page Free : O(1)

Size of mm data structures

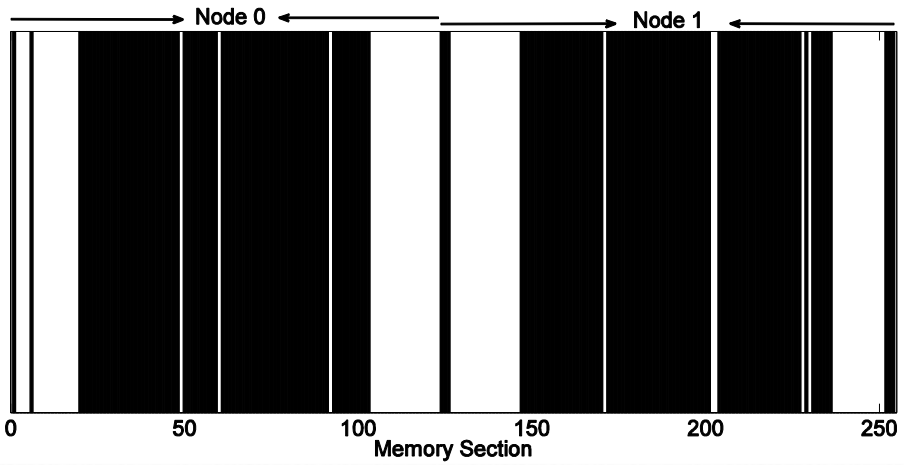


Page Migration

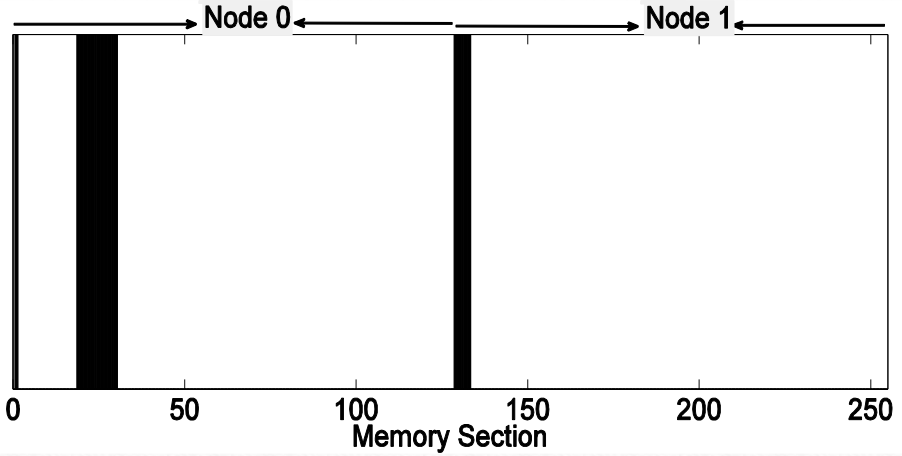
- Sort banks in descending order based on the number of free pages.
- Identify candidate banks for migration (break point).
- Isolate candidate banks from allocation pool (to avoid copying more than once).
- Migrate in the sorted order.



Results



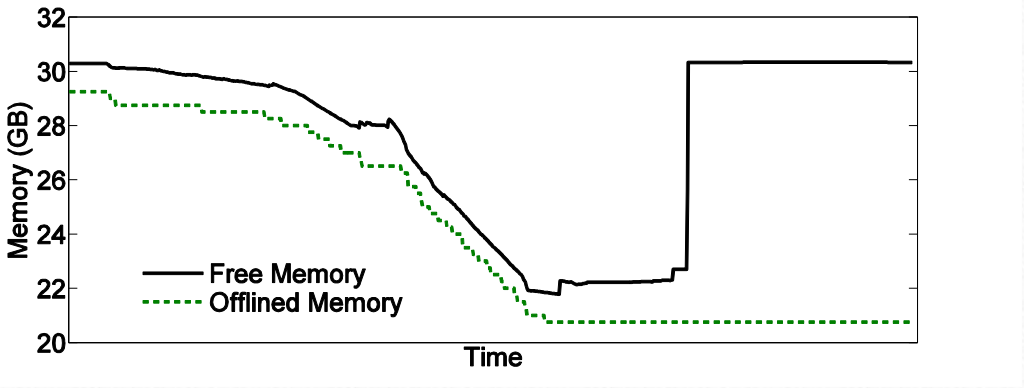
(a). Unmovable memory sections (colored black) with Buddy allocator



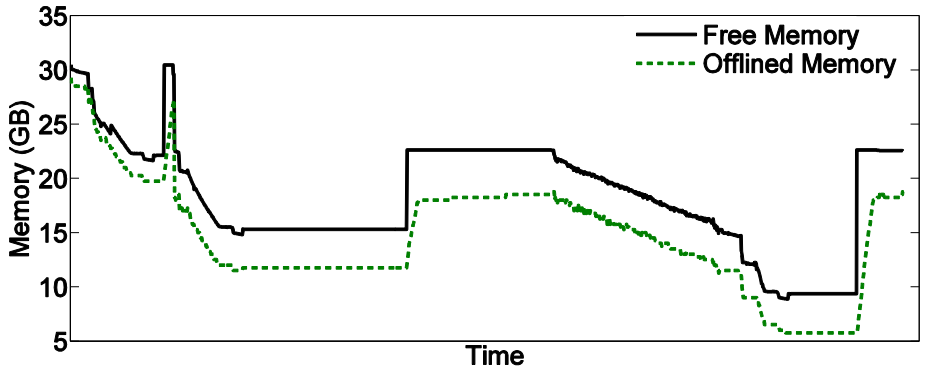
(b). Unmovable memory sections (colored black) with Adaptive-Buddy allocator

Fig 9. Kernel memory footprint (sections colored black) with different allocators

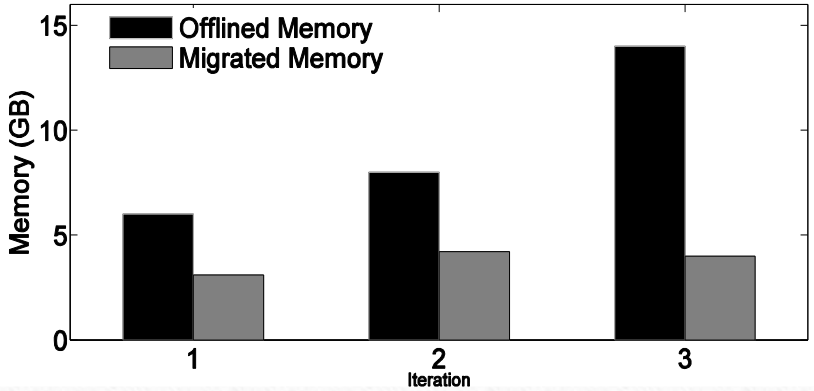
Results



(a). Free vs Offlined memory without page migration



(b). Free vs Offlined memory with page migration



(c). Offlined vs migrated Memory

Fig 10. Behavior of Adaptive-Buddy regarding power management

Results

Workloads –
Light : Memory footprint varying between 10%-50%.
Medium : Memory footprint varying between 30%-80%.

Workload	32 GB			4 GB		
	Max	Average Offline Memory(%)		Max	Average Offline Memory(%)	
		Optimal	Adaptive-Buddy		Optimal	Adaptive-Buddy
Light	81	69	59.5	64	55	48
Medium	56	57	45.7	45	35	26

Table 2. Percentage of offline memory on different systems and workloads along with comparison with a hypothetical optimal solution in the average case.

Performance Evaluation

PARSEC Benchmark Suite

Application	User Time (seconds)		System Time (ms)	
	Buddy	Adaptive-Buddy	Buddy	Adaptive-Buddy
bodytrack	290	290	864	672
blackscholes	352	352	3885	3915
canneal	297	299	7380	7412
dedup	56	55	27843	28363
facesim	1051	1058	9610	9836
freqmine	987	981	2360	2382
streamcluster	1052	1094	12319	12614
swaptions	666	639	13373	12228
vips	273	272	4478	3848
x264	211	211	1603	1787

Table 3. Comparison between the execution time with different allocators for PARSEC applications

Performance Evaluation

AIM9 Microbenchmark Suite

Test	Description	Operations per Second		Difference
		Buddy	Adaptive-Buddy	
creat-clo	File creations & closes	151816	**149883**	**-1.30%**
page_test	System allocations & pages	552330	**559781**	**+1.35%**
break_test	System memory allocations	3716713	**3949383**	**+6.26%**
exec_test	Program loads	1396	1390	-0.43%
fork_test	Task creations	3260	**3193**	**-2.05%**
shared_memory	Shared memory operations	1054128	1053623	-0.05%
tcp_test	TCP/IP messages	202990	202461	-0.26%
udp_test	UDP/IP datagrams	458661	**449836**	**-1.93%**

Table 4. Comparison between the execution time with different allocators for PARSEC applications

Performance Evaluation

A Synthetic Benchmark:

- Created for testing the page allocation performance for a large number of successive requests to build memory pressure.
- *C* program for page allocation and *systemtap* for timing measurement.

No. of Pages	Size	Average (ns)		Worst(ns)	
		Buddy	Adaptive-Buddy	Buddy	Adaptive-Buddy
1 x 10^4	40 MB	285	267	755	1305
1 x 10^5	400 MB	266	260	988	2596
1 x 10^6	4 GB	260	255	999	20794
2 x 10^6	8 GB	250	255	999	20794
4 x 10^6	16 GB	249	277	999	21432

Table 5. Comparison between the average and worst case allocation time of different allocators

- Worst case behavior of Adaptive-Buddy can be improved by using a kernel thread for moving pages of a memory bank from one pool to another but that solution also has many downsides !

Future Work

- Integrating and testing Adaptive-Buddy on a real hardware.
- Evaluating Adaptive-Buddy in virtualized environments (for capacity on demand).
- A power friendly buffer cache ?

References

- https://lwn.net/Articles/478049/
- Linux Memory Power Management [S. Bhat 2013].
- Linux VM Infrastructure to support Memory Power Management [A. Garg 2011].
- Jantz, Michael R. and Strickland, Carl and Kumar, Karthik and Dimitrov, Martin and Doshi, ``A Framework for Application Guidance in Virtual Memory Systems", VEE 2013.
- David, Howard and Fallin, Chris and Gorbatov, Eugene and Hanebutte, Ulf R. and Mutlu, Onur, ``Memory Power Management via Dynamic Voltage/Frequency Scaling", ICAC 2011.
- Huang, Hai and Shin, Kang G. and Lefurgy, Charles and Keller, Tom, ``Improving Energy Efficiency by Making DRAM Less Randomly Accessed", ISPLED 2005.
- Huang, Hai and Pillai, Padmanabhan and Shin, Kang G., ``Design and implementation of power-aware virtual memory", ATEC 2003.
- Hai Huang, Kang G. Shin, Charles Lefurgy, Karthick Rajamani, Tom Keller, Eric Van Hensbergen, and Freeman Rawson, "Cooperative Software-Hardware Power Management for Main Memory", published in Lecture Notes in Computer Science, *Power-Aware Computer Systems: 4th International Workshop, PACS 2004*, Volume 3471, December, 2005, pages 61-77.