Using Umpire's Coalescing Heuristics to Improve Memory Performance

Kristi Belcher, David Beckingsale, Marty McFadden

Lawrence Livermore National Laboratory, Livermore, CA, USA

The performance of memory pools in scientific applications varies widely depending upon how blocks of memory within the pool are managed. The Umpire team from Lawrence Livermore National Laboratory (LLNL) studied a high-explosive chemistry application and conducted experiments to study different ways to manage blocks of memory within the pool. Our study demonstrates that with the right heuristic we can see memory savings up to 64% which, for this code, translated to an 8-16x speedup.

Background

- Umpire (developed at LLNL) provides memory pools which allow a less expensive way to allocate all needed memory for HPC applications, compared to device specific APIs.

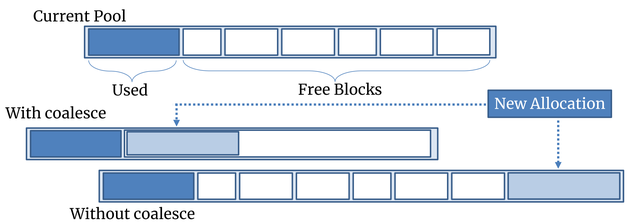

Figure 1: When a new allocation is added to an Umpire pool, it will either readjust to accommodate it (with coalescing) or simply grow larger (without coalescing).

- To handle new allocations, a coalesce function deallocates any unused blocks in the pool and reallocates one large block instead (Fig. 1).
- Umpire's heuristics help tune the coalesce function to better manage the blocks within a memory pool.

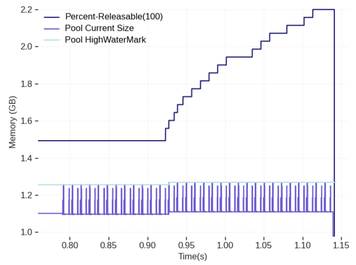

Figure 2: The memory pool, using the previous Percent-Releasable heuristic, grew too large and caused the chemistry application to crash, running out of memory prematurely.

- If the pool can't coalesce successfully, it can grow too large (Fig. 2).
- The coalesce function can be an expensive operation because it involves memory (de/re)allocation.

Experiments

Experiments performed by the Umpire Team to test how different coalescing heuristics affect Umpire's memory pool performance:

1. **Bytes-Based (*Percent-Releasable*) heuristic:** once a certain percentage of bytes is releasable, coalesce the pool. Testing 50%, 75%, 90%, and 100% of bytes.
2. **Blocks-Based (*Blocks-Releasable*) heuristic:** once a certain number of blocks is releasable, coalesce the pool. Testing 2, 3, 5, 7, and 12 blocks.
3. **High-WaterMark (*HWM*) heuristic tuning:** when a coalesce is needed, coalesce to the high-watermark instead of the actual size of the pool.
4. **Coalesce-Before-Growing (*CBG*) heuristic tuning:** check to see if a coalesce is needed as the pool grows instead of after deallocation.

Results

Heuristic Name	Coalesce Calls	Memory Overhead
Percent(50)	*Timed Out*	*Timed Out*
Percent(75)	*Timed Out*	*Timed Out*
Percent(90)	*Timed Out*	*Timed Out*
Percent(100)	2	73%
Blocks(2)	44	8%
Blocks(3)	30	**6%**
Blocks(5)	18	15%
Blocks(7)	11	11%
Blocks(12)	**7**	17%
Blocks(2) HWM	47	7%
Blocks(3) HWM	34	**5%**
Blocks(5) HWM	17	8%
Blocks(7) HWM	12	11%
Blocks(12) HWM	**9**	18%
Blocks(2) CBG	10	**10%**
Blocks(3) CBG	8	15%
Blocks(5) CBG	7	19%
Blocks(7) CBG	5	13%
Blocks(12) CBG	**4**	26%

Table 1: Number of coalesce calls compared to the *percentage of memory overhead* shows a tradeoff for a variety of Blocks-Releasable heuristic functions. Most Percent-Releasable heuristics resulted in time outs.

- The **Percent-Releasable (100%) heuristic was worst** at 73% memory overhead (Table 1).

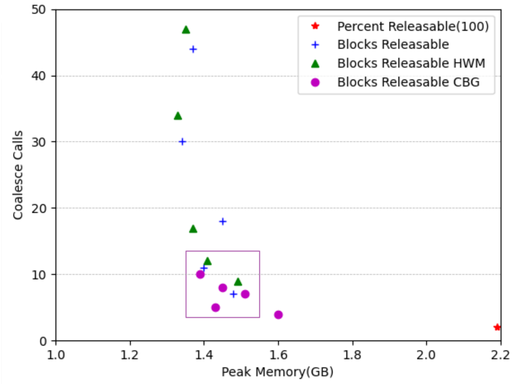

Figure 3: The overall trends with the number of coalesce calls plotted against the *peak memory usage* remains consistent throughout all Blocks-Releasable experiments. The box represents a sweet spot range.

- As the number of blocks required to coalesce increases from 2 to 12, the number of total coalesce calls decreases and the peak memory usage (memory overhead) increases (Fig. 3).
- The **sweet spot** is a range of values that is not too expensive in terms of both the number of coalescing calls and total memory used.

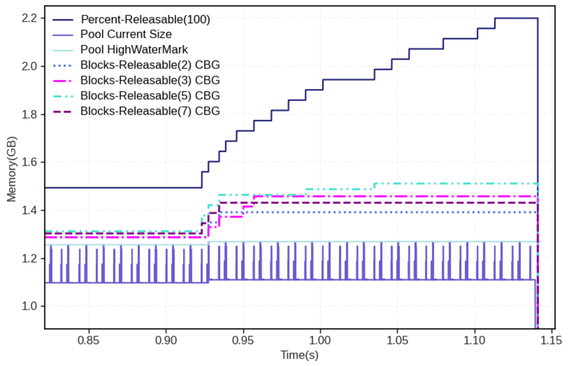

Figure 4: The amount of memory used with the Blocks-Releasable heuristic experiments tuned at Coalesce-Before-Grow (dashed/dotted lines) compared to the Percent-Releasable heuristic from Fig. 2 (solid line) shows a big improvement.

- The **Blocks-Releasable heuristic with the Coalesce-Before-Grow tuning represented the best results** for this application.
- From all our Blocks-Releasable experimental results, we saw a **36-64% reduction in total memory usage** (Fig. 4).
- These memory savings translated to **8-16x speedup** for this application.

Learn More!

Scan the QR code to see our GitHub branch with experiments and reproducibility instructions.

Clone Umpire:
https://github.com/LLNL/Umpire

Contact the Umpire Team:
umpire-dev@llnl.gov

Conclusions and Future Work

- Effectively managing blocks of memory within a memory pool with Umpire's coalescing heuristics can dramatically improve overall memory pool performance.
- The **Blocks-Releasable heuristic worked better** because it was able to successfully trigger the coalesce function when the memory pool needed to readjust.
- After applying the Blocks-Releasable heuristic to our chemistry application, it used **36-64% less total memory and had an 8-16x speedup**.
- Unused blocks of memory within a pool are much less likely to occur if different types of memory allocations are separated into distinct pools.
- Future work will include further study into how coalescing functions impact other application codes.
- Additional future work involves studying ways to separate different types (temporary and permanent) of allocations automatically using decision models.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory.

LLNL-POST-838469