mallocMC

Bernhard Manfed Gruber

A Memory Allocator for Many Core Architectures

- Primarily intended for CUDA
- CUDA C provides:
  - void* ::malloc(size_t size);
  - void ::free(void* ptr);
- ::malloc() allocates from a global heap in GPU memory
  - High contention when allocation from many threads (possibly 1000s)
  - Slow

ScatterAlloc - Paper

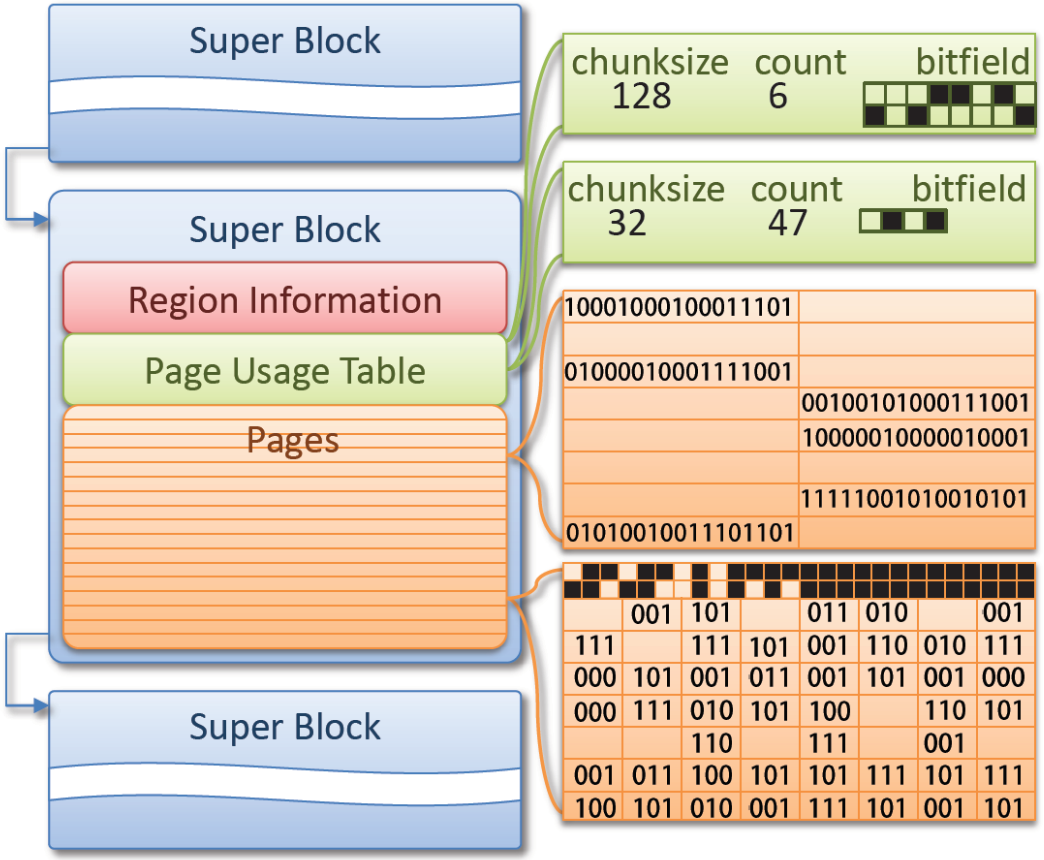
- **ScatterAlloc: Massively Parallel Dynamic Memory Allocation for the GPU**
- Markus Steinberger, Michael Kenzel, Bernhard Kainz, Dieter Schmalstieg
- Institute for Computer Graphics and Vision Graz University of Technology
- 2012

ScatterAlloc - Paper

- Defines a lot of useful requirements for GPU allocators
    - Correctness, memory access performance, scalabilty, branch divergence, false sharing, coalesced access, internal/external fragmentation, blowup
- Assumes allocations are for small objects and often for the same size
- General allocation overhead stays below 1%
- 100x faster than native CUDA ::malloc()

ScatterAlloc - Paper

- Memory split into superblocks, superblocks split into pages
- Pages managed by a page usage table inside the superblock
- Superblocks have meta data for finding free regions quickly
- Pages are split into chunks, with different chunk sizes
- Hashing (SMP, alloc size) to find a page for allocation
- Allocations larger than page: dedicated superblock



mallocMC

- An implementation of ScatterAlloc for CUDA/Alpaka
- https://github.com/ComputationalRadiationPhysics/mallocMC
- Header only library
- Policy-based design
  - To be extendable for additional allocation algorithms
- Template heavy

Policies

- Alignment policy
  - Aligns pool and chunk sizes during malloc()
  - Noop, Shrink
- Creation policy
  - Allocates a piece of memory in the pool, serves malloc() requests
  - OldMalloc, Scatter
- Distribution policy
  - Unify allocations requests and redistribute memory chunk
  - Noop, XMallocSIMD
- OOM policy
  - Handle out-of-memory condition
  - BadAllocException, ReturnNull
- Reserve pool policy
  - How the memory pool is provided
  - AlpakaBuf, CudaSetLimits

Alpaka port

- MallocMC is currently ported to alpaka
  - Alpaka is a library for parallel kernel acceleration
  - My focus of the last few weeks
- Opens up mallocMC from CUDA to more computing technologies
  - HIP, OpenMP, OpenAcc, std::thread
- Introduces dependency on alpaka for kernel invocation

Example

```
struct ScatterHeapConfig {
    static constexpr auto pagesize        = 4096;
    static constexpr auto accessblocks    = 8;
    static constexpr auto regionsize      = 16;
    static constexpr auto wastefactor     = 2;
    static constexpr auto resetfreedpages = false;
};
struct ScatterHashConfig {
    static constexpr auto hashingK         = 38183;
    static constexpr auto hashingDistMP    = 17497;
    static constexpr auto hashingDistWP    = 1;
    static constexpr auto hashingDistWPRel = 1;
};
struct XMallocConfig {
    static constexpr auto pagesize = ScatterHeapConfig::pagesize;
};
struct ShrinkConfig {
    static constexpr auto dataAlignment = 16;
};
```

Example

```
using Dim = alpaka::dim::DimInt<1>;
using Idx = size_t;
using Acc = alpaka::acc::AccGpuCudaRt<Dim, Idx>;

using ScatterAllocator = mallocMC::Allocator<
    Acc,
    mallocMC::CreationPolicies::Scatter<ScatterHeapConfig, ScatterHashConfig>,
    mallocMC::DistributionPolicies::XMallocSIMD<XMallocConfig>,
    mallocMC::OOMPolicies::ReturnNull,
    mallocMC::AlignmentPolicies::Shrink<ShrinkConfig>
>;
```

Example

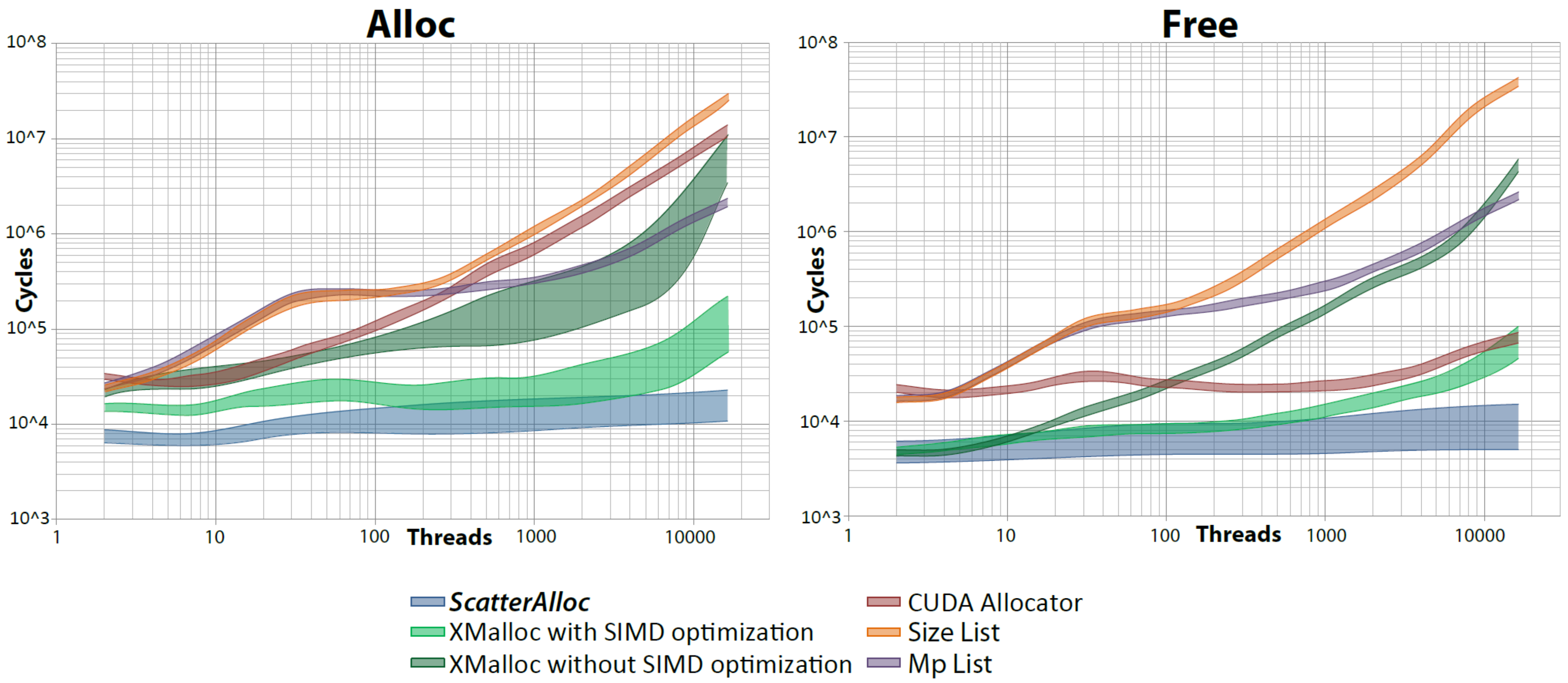
```
const auto dev
    = alpaka::pltf::getDevByIdx<Acc>(0);
auto queue = alpaka::queue::Queue<Acc,
    alpaka::queue::Blocking>{dev};

ScatterAllocator scatterAlloc{dev, queue,
    1U * 1024U * 1024U * 1024U}; // 1GB
```

Example

```
auto kernelFunc = [] ALPAKA_FN_ACC(
    const Acc& acc,
    ScatterAllocator::AllocatorHandle handle) {
        int* mem = (int*)handle.malloc(acc, sizeof(int) * 100);
        // ...
        handle.free(acc, mem);
};
const auto workDiv = ...;
alpaka::queue::enqueue(
    queue, alpaka::kernel::createTaskKernel<Acc>(
        workDiv, kernelFunc, scatterAlloc.getAllocatorHandle()
    )
);
```

Benchmark



Number of clock cycles needed to serve a single alloc or free for 16, 32, 64, and 128 bytes.