A Modern Memory Management System for OpenMP

J. D. Sewall, **S. J. Pennycook**, A. Duran, X. Tian and R. Narayanaswamy
Intel Corporation

Workshop on Accelerator Programming Using Directives (WACCPD) 2016

Motivation: Variety in Memory Hierarchies

Platform	Memory Kind							
	Constant	Texture	SPM	DDR	eDRAM	GDDR	HBM	NVRAM
Intel® Xeon® Processor	-	-	-	✓	-	-	-	-
Intel® Xeon Phi™ Coprocessor	-	-	-	-	-	✓	-	-
Intel® Xeon Phi™ Processor	-	-	-	✓	-	-	✓	-
Future System w/ 3D XPoint™ Technology	-	-	-	✓	-	-	-	✓
Intel® HD Graphics	-	-	✓	✓	✓	-	-	-
Intel® Iris™ Graphics	-	-	✓	✓	✓	-	-	-
Current Generation NVIDIA* GPU	✓	✓	✓	-	-	✓	-	-
Future Generation NVIDIA* GPU	✓	✓	✓	-	-	✓	✓	-

Motivation: Inadequate Allocation Interfaces

- Language standards are not keeping pace with modern hardware:
  - Legacy `malloc`, `new`/`delete`, `ALLOCATE` take one parameter: size.
  - Later additions have focused primarily on alignment.

- A growing number of incompatible and overlapping options in development:
  - Proprietary allocators (e.g. `_mm_malloc`, `cudaMalloc`)
  - Scalable threaded allocators (e.g. `jemalloc`, `TBBmalloc`)
  - OS functionality (e.g. `LD_PRELOAD`, `numactl`)
  - Environment variables (e.g. `MKL_FAST_MEMORY_LIMIT`)

Design: Key Principles

- Three driving principles:
  1. Support as many existing memories as possible.
  2. Support future memories without requiring significant changes.
  3. Support all types of user allocations (e.g. static, stack, heap).

- We believe that a successful interface for HPC should also:
  - Be simple to integrate into large codes.
  - Be aware of (and make allowances for) legacy interfaces that cannot be changed.
  - Be compatible with existing HPC programming models.

Proposal: Key Concepts

- **Traits**
  - Descriptive characteristics that can be queried and specified by the user.
- **Memory Space**
  - A particular system-level storage resource.
  - Example Traits: “Kind”; Page Size; Permissions; Persistence; Capacity
- **Allocator**
  - An object that manages memory allocations from a given memory space.
  - Example Traits: Thread Safety; Default Alignment; Pinning; Fallback Behavior

Proposal: Example Usage of the API

```
omp_memtrait_t lbm_traits[] = { {OMP_MTK_BANDWIDTH, OMP_MTV_LOWEST}, {OMP_MT_PAGESIZE, 2*1024*1024} };
omp_memtrait_t hbm_traits[] = { {OMP_MTK_BANDWIDTH, OMP_MTV_HIGHEST}, {OMP_MT_PAGESIZE, 2*1024*1024} };

omp_memtrait_set_t lbmset, hbmset;

omp_init_memtrait_set(&lbmset, 2, lbm_traits);
omp_init_memtrait_set(&hbmset, 2, hbm_traits);

omp_memspace_t* ddr_mem = omp_init_memspace(&lbmset);
omp_memspace_t* hbw_mem = omp_init_memspace(&hbmset);

omp_alloctrait_set smallset, largeset;

omp_alloctrait_t small_traits[] = { {OMP_ATK_ALIGNMENT, 64}, {OMP_ATK_FALLBACK, OMP_ATV_ABORT} };
omp_init_alloctrait_set(&smallset, 2, small_traits);
omp_allocator_t* small_allocator = omp_init_allocator(ddr_mem, &smallset);

omp_alloctrait_t large_traits[] = { {OMP_ATK_ALIGNMENT, 64}, {OMP_ATK_FALLBACK, OMP_ATV_ALLOCATOR},
                                    {OMP_ATK_FBDATA, small_allocator} };
omp_init_alloctrait_set(&largeset, 3, large_traits);
omp_allocator_t* large_allocator = omp_init_allocator(hbw_mem, &largeset);

…

void foo(omp_allocator_t* allocator)
{
        double* array = (double*) omp_allocate(allocator, sizeof(double)*N);
        …
        omp_free(allocator, array);
}
```

Key
type
trait
API

Proposal: Example Usage of the Directives/Clauses

```
double a[N];
#pragma omp allocate(a) memtraits(bandwidth=lowest, pagesize=2*1024*1024) // allocate directive

void foo(omp_allocator_t* allocator)
{
      double b[N];
      #pragma omp allocate(b) allocator(allocator)

      double c[M];
      #pragma omp parallel firstprivate(b) private(c) \
      allocate(memtraits(bandwidth=highest, pagesize=2*1024*1024):b,c) // allocate clause
      {
            …
      } // private copies of c are automatically deallocated at the exit of this scope

} // b and c are automatically deallocated at the exit of this scope
```

Key
type
trait
directive
clause

Proposal: Support for Special Instructions

```
#pragma omp declare version(foo_persistent) memtraits(persistence=true:v)
#pragma omp declare version(foo_scratch) memtraits(location=core:v)
void foo(double* v) { … }

#pragma omp declare version implements(foo) memtraits(optimized=latency:v)
void foo_fancy(double* v) { … }

void bar (double* a)
{
      double b[N];
      #pragma omp allocate(b) memtraits(location=core)

      #pragma omp dispatch(b)
      foo(b); // compiler can see static memtraits of ‘b’ so can call foo_scratch

      #pragma omp dispatch(a)
      foo(a); // compiler must perform reflection on ‘a’ for dynamic dispatch

      foo_fancy(a); // user can call foo_fancy manually, based on program knowledge
}
```

Key
type
trait
directive

Summary

- We have proposed a novel mechanism for memory management that:
  - Separates **storage resources** from **allocation behavior**.
  - Provides a **platform-agnostic interface** for querying and managing memory.
  - Is **compatible with OpenMP*** directives.

- Working on an OpenMP TR with newest candidate directives/API for release by end of 2016.

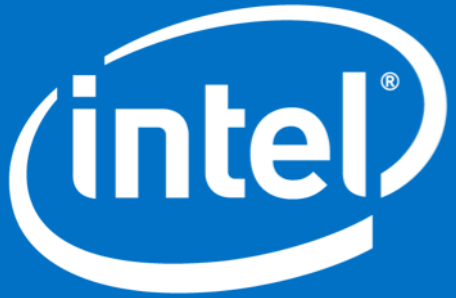
- Future work:
  - Continue to refine and iterate over our proposal until it's accepted by the standard. ☺

Legal Notices and Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Intel, Xeon, Xeon Phi, 3D XPoint, Iris, the Intel logo and others are trademarks of Intel Corporation in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others.


intel