



DE LA RECHERCHE À L'INDUSTRIE

Preliminary Experience with OpenMP Memory Management Implementation

Adrien Roussel^{1,2,3}, **Patrick Carribault**^{1,3}, **Julien Jaeger**^{1,2,3}

International Workshop on OpenMP | Sept. 21-24 2020

¹ CEA, DAM, DIF, F-91297 Arpajon, France

² Exascale Computing Research

³ Laboratoire en Informatique Haute Performance pour le Calcul et la simulation

- Main evolution for the past decades
 - Frequency increase
 - Cores number increase
 - Functional unit extension (e.g., SIMD)

- Main evolution for the past decades
 - Frequency increase
 - Cores number increase
 - Functional unit extension (e.g., SIMD)
- Stress is now focused on the memory part

- Main evolution for the past decades
 - Frequency increase
 - Cores number increase
 - Functional unit extension (e.g., SIMD)
 - Stress is now focused on the memory part
-
- ⇒ New memory types appear
- High Bandwidth Memory (HBM)
 - MCDRAM (Intel KNL processor)
 - HBM (Fujitsu A64FX processor)
 - Non-volatile memory
 - NVDIMM

- Memory allocator library

- glibc
- Memkind (Intel)
- VMEM (Intel)
- ...

- + Fine control on data allocation

- One allocator per memory space
- Highly intrusive solution in codes

- Memory allocator library

- glibc
- Memkind (Intel)
- VMEM (Intel)
- ...

+ Fine control on data allocation

- One allocator per memory space
- Highly intrusive solution in codes

- High level abstractions

- Programming interface
- Domain Specific Language

+ Simple & transparent for users

+ Performance portability

- No control on data allocation process

- Memory allocator library

- glibc
 - Memkind (Intel)
 - VMEM (Intel)
 - ...

- + Fine control on data allocation

- One allocator per memory space
 - Highly intrusive solution in codes

- High level abstractions

- Programming interface
 - Domain Specific Language

- + Simple & transparent for users

- + Performance portability
 - No control on data allocation process

→ No standard way to handle memory management inside applications

- Memory allocator library

- glibc
 - Memkind (Intel)
 - VMEM (Intel)
 - ...

- + Fine control on data allocation

- One allocator per memory space
 - Highly intrusive solution in codes

- High level abstractions

- Programming interface
 - Domain Specific Language

- + Simple & transparent for users

- + Performance portability
 - No control on data allocation process

→ No standard way to handle memory management inside applications

⇒ Memory management constructs into OpenMP 5.0

1 Introduction to OpenMP Memory Management

- Memory Management Extensions

2 Memory Management Integration into OpenMP runtime

- Hardware Detection
- Memory Allocator Initialization Process
- Data Allocation/Deallocation Process

3 Implementation into the MPC framework**4** Integrating OpenMP Memory Management constructs in Software

- LULESH Application
- Memory Management Support in C++ Application

5 Evaluation**6** Conclusion and Future work

1 Introduction to OpenMP Memory Management

- Memory Management Extensions

2 Memory Management Integration into OpenMP runtime

- Hardware Detection
- Memory Allocator Initialization Process
- Data Allocation/Deallocation Process

3 Implementation into the MPC framework**4** Integrating OpenMP Memory Management constructs in Software

- LULESH Application
- Memory Management Support in C++ Application

5 Evaluation**6** Conclusion and Future work

```
float A[N], B[N];
#pragma omp allocate(A) \
    allocator(omp_large_cap_mem_alloc)

#pragma omp parallel
{
    #pragma omp task
    {
        /* ... */
    }
}
```

■ allocate directive

- How a set of variables are allocated

■ Information needed

```
float A[N], B[N];
#pragma omp allocate(A) \
    allocator(omp_large_cap_mem_alloc)

#pragma omp parallel
{
    #pragma omp task
    {
        /* ... */
    }
}
```

■ allocate directive

- How a set of variables are allocated

■ Information needed

- List of variables

```
float A[N], B[N];
#pragma omp allocate(A) \
    allocator(omp_large_cap_mem_alloc)

#pragma omp parallel
{
    #pragma omp task
    {
        /* ... */
    }
}
```

■ allocate directive

- How a set of variables are allocated

■ Information needed

- List of variables
- Allocator among pre-defined ones

Pre-defined allocators

- `omp_default_mem_alloc`
- `omp_large_cap_mem_alloc`
- `omp_const_mem_alloc`
- `omp_high_bw_mem_alloc`
- `omp_low_lat_mem_alloc`
- `omp_cgroup_mem_alloc`
- `omp_pteam_mem_alloc`
- `omp_thread_mem_alloc`

```
float A[N], B[N];
#pragma omp allocate(A) \
    allocator(omp_large_cap_mem_alloc)

#pragma omp parallel
{
    #pragma omp task private(B) allocate(omp_const_mem_alloc: B)
    {
        /* ... */
    }
}
```

- **allocate clause**
 - Memory allocator to use for **private** variables

```
#define N 1000

int main()
{
    float *x, *y;

    omp_memspace_handle_t memspace = omp_default_mem_space;
    omp_allocator_traits_t traits[1] = {omp_atk_alignment, 64};
    omp_allocator_handle_t alloc = omp_init_allocator(x_memspace, 1, traits);

    x = (float*) omp_alloc(N * sizeof(float), alloc);

    #pragma omp parallel
    {
        /* Do some computations here */
    }

    omp_free(x, alloc);

    omp_destroy_allocator(alloc);

    return 0;
}
```

```
#define N 1000

int main()
{
    float *x, *y;

    omp_memspace_handle_t memspace = omp_default_mem_space;
    omp_allocator_traits_t traits[1] = {omp_atk_alignment, 64};
    omp_allocator_handle_t alloc = omp_init_allocator(x_memspace, 1, traits); } — Init allocator

    x = (float*) omp_alloc(N * sizeof(float), alloc);

    #pragma omp parallel
    {
        /* Do some computations here */
    }

    omp_free(x, alloc);

    omp_destroy_allocator(alloc);

    return 0;
}
```

```
#define N 1000

int main()
{
    float *x, *y;

    omp_memspace_handle_t memspace = omp_default_mem_space;
    omp_allocator_traits_t traits[1] = {omp_atk_alignment, 64};
    omp_allocator_handle_t alloc = omp_init_allocator(x_memspace, 1, traits); } — Init allocator

    x = (float*) omp_alloc(N * sizeof(float), alloc); } — Allocation

#pragma omp parallel
{
    /* Do some computations here */
}

    omp_free(x, alloc);

    omp_destroy_allocator(alloc);

    return 0;
}
```

```
#define N 1000

int main()
{
    float *x, *y;

    omp_memspace_handle_t memspace = omp_default_mem_space;
    omp_allocator_traits_t traits[1] = {omp_atk_alignment, 64};
    omp_allocator_handle_t alloc = omp_init_allocator(x_memspace, 1, traits); } — Init allocator

    x = (float*) omp_alloc(N * sizeof(float), alloc); } — Allocation

#pragma omp parallel
{
    /* Do some computations here */
}

    omp_free(x, alloc); } — Deallocation

    omp_destroy_allocator(alloc);

    return 0;
}
```

```
#define N 1000

int main()
{
    float *x, *y;

    omp_memspace_handle_t memspace = omp_default_mem_space;
    omp_allocator_traits_t traits[1] = {omp_atk_alignment, 64};
    omp_allocator_handle_t alloc = omp_init_allocator(x_memspace, 1, traits); } — Init allocator

    x = (float*) omp_alloc(N * sizeof(float), alloc); } — Allocation

#pragma omp parallel
{
    /* Do some computations here */
}

    omp_free(x, alloc); } — Deallocation
    omp_destroy_allocator(alloc); } — Release allocator

    return 0;
}
```

1 Introduction to OpenMP Memory Management

- Memory Management Extensions

2 Memory Management Integration into OpenMP runtime

- Hardware Detection
- Memory Allocator Initialization Process
- Data Allocation/Deallocation Process

3 Implementation into the MPC framework**4** Integrating OpenMP Memory Management constructs in Software

- LULESH Application
- Memory Management Support in C++ Application

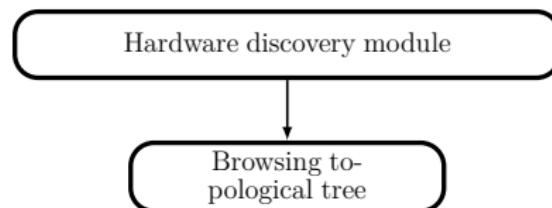
5 Evaluation**6** Conclusion and Future work

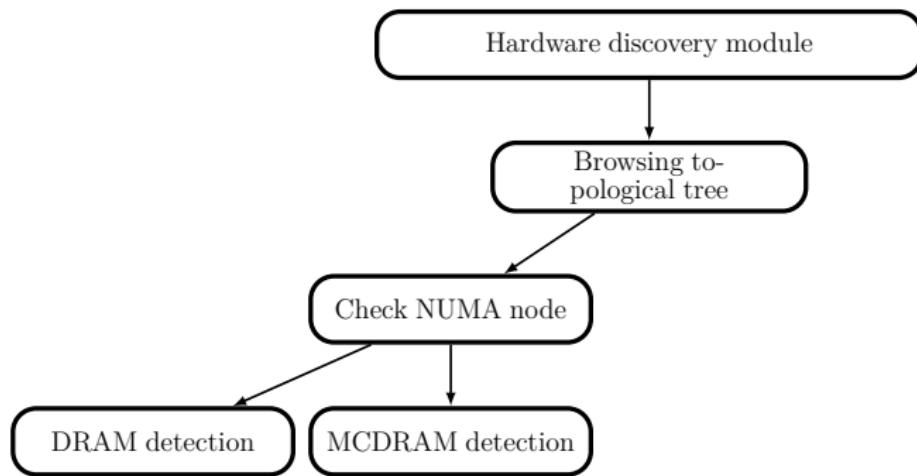
- Various memory types are supported in OpenMP

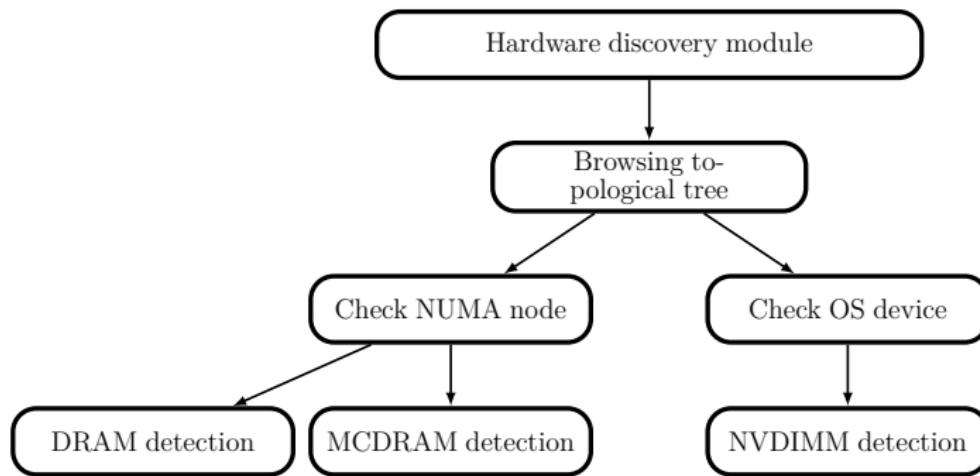
- Various memory types are supported in OpenMP
- However...
 - They are not available on all machines!

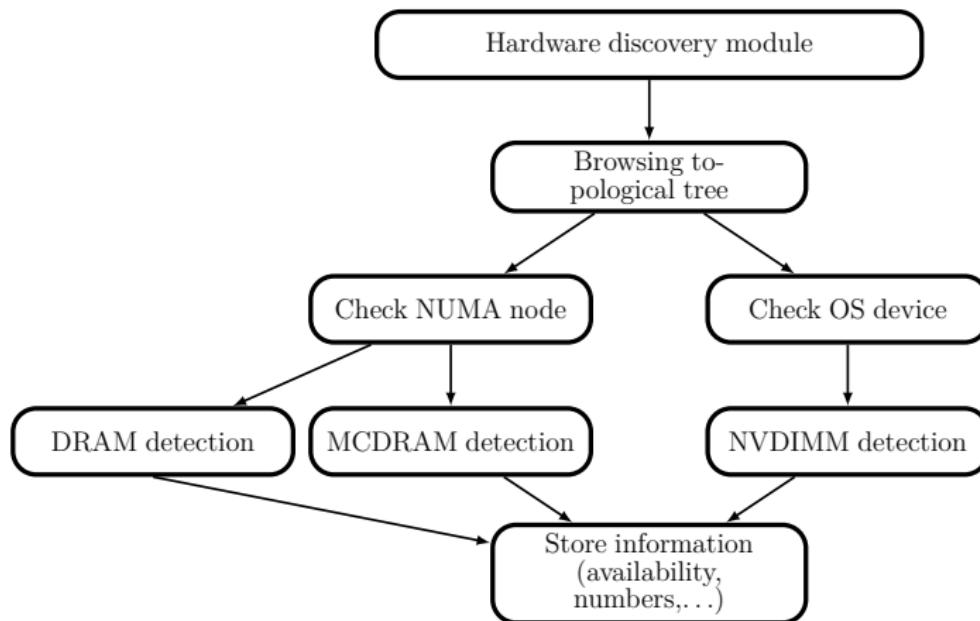
- Various memory types are supported in OpenMP
- However...
 - They are not available on all machines!
- Need information about available memory spaces
 - Adding a memory-type detection module

Hardware discovery module









- Memory spaces

- `omp_default_mem_space`
- `omp_large_cap_mem_space`
- `omp_const_mem_space`
- `omp_high_bw_mem_space`
- `omp_low_lat_mem_space`

- Binding

- OpenMP memory space
- Hardware



Implementation defined

- Memory spaces

- `omp_default_mem_space = DRAM`
- `omp_large_cap_mem_space = NVDIMM`
- `omp_const_mem_space = DRAM`
- `omp_high_bw_mem_space = MCDRAM`
- `omp_low_lat_mem_space = DRAM`

- Binding

- OpenMP memory space
- Hardware

{}

Implementation defined

- Make choices!

- Memory spaces

- `omp_default_mem_space = DRAM`
- `omp_large_cap_mem_space = NVDIMM`
- `omp_const_mem_space = DRAM`
- `omp_high_bw_mem_space = MCDRAM`
- `omp_low_lat_mem_space = DRAM`

- Binding

- OpenMP memory space
- Hardware

{

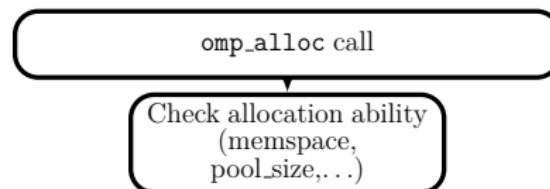
Implementation defined

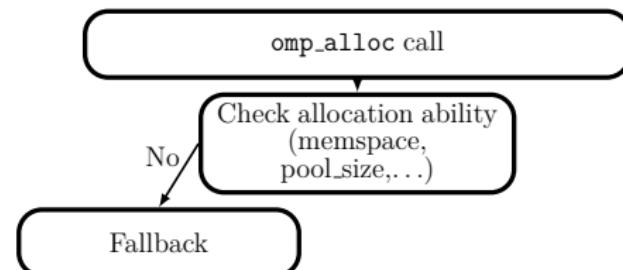
- Make choices!

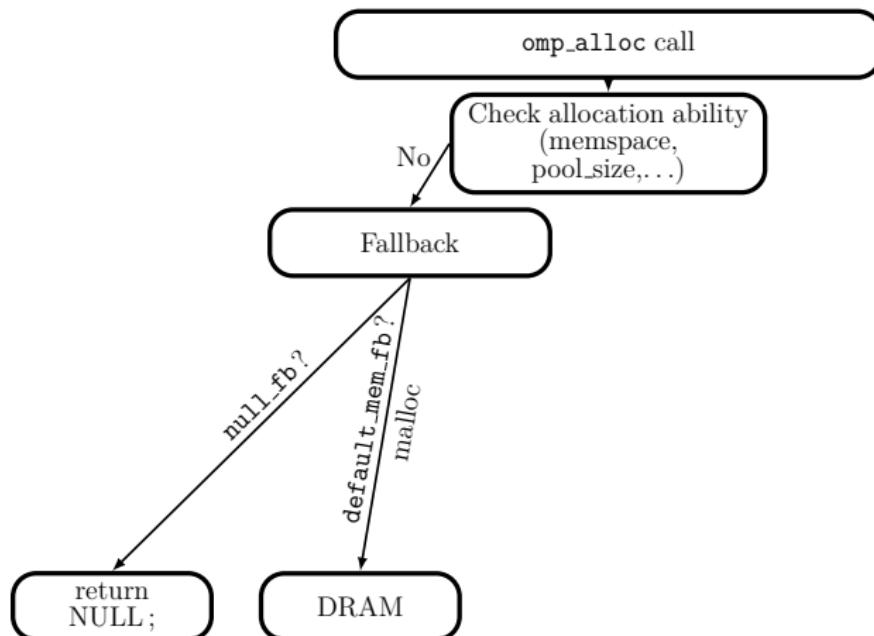
- If a memory type is not detected
 - Set to default memory space

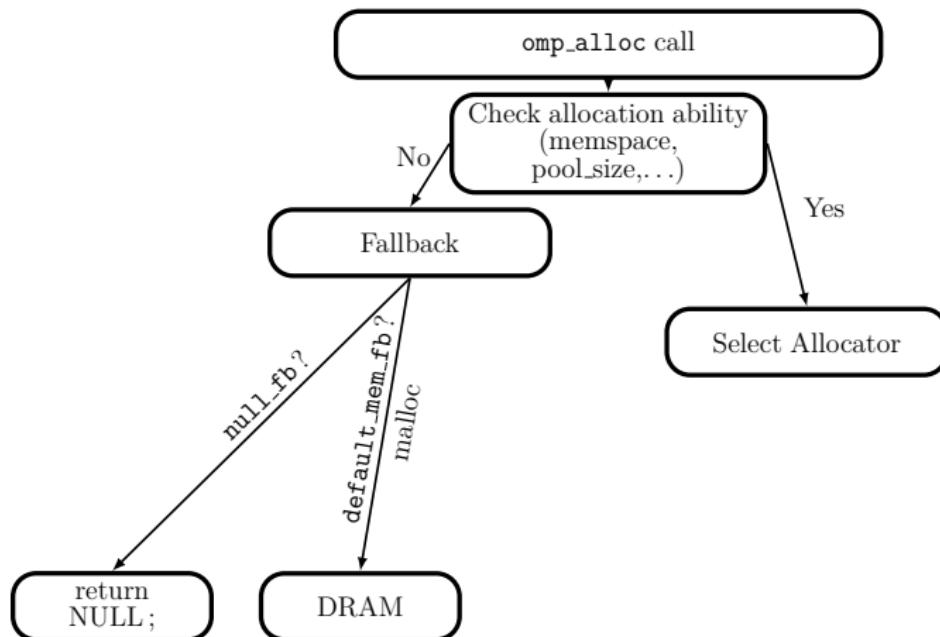
- `omp_alloc` function
 - User requests n bytes to be allocated in the memory space referred by `omp_mem_space_handle_t`
- Data allocations into various memory spaces
 - ⇒ Integration of memory allocator libraries
 - Examples:
 - glibc malloc for DRAM
 - memkind library for high bandwidth and large capacity memories
 - vmem for large capacity memory
 - ...

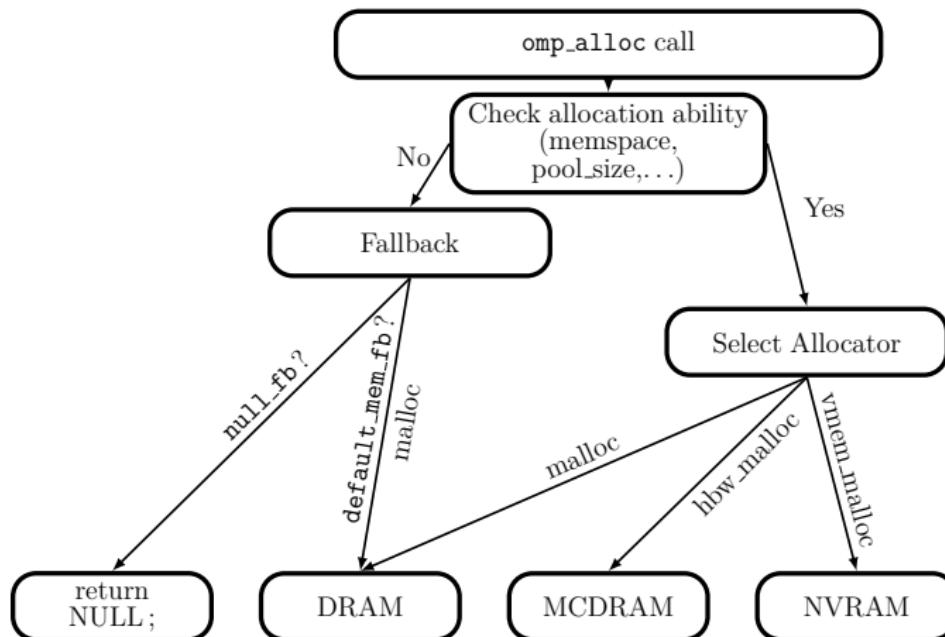
omp_alloc call

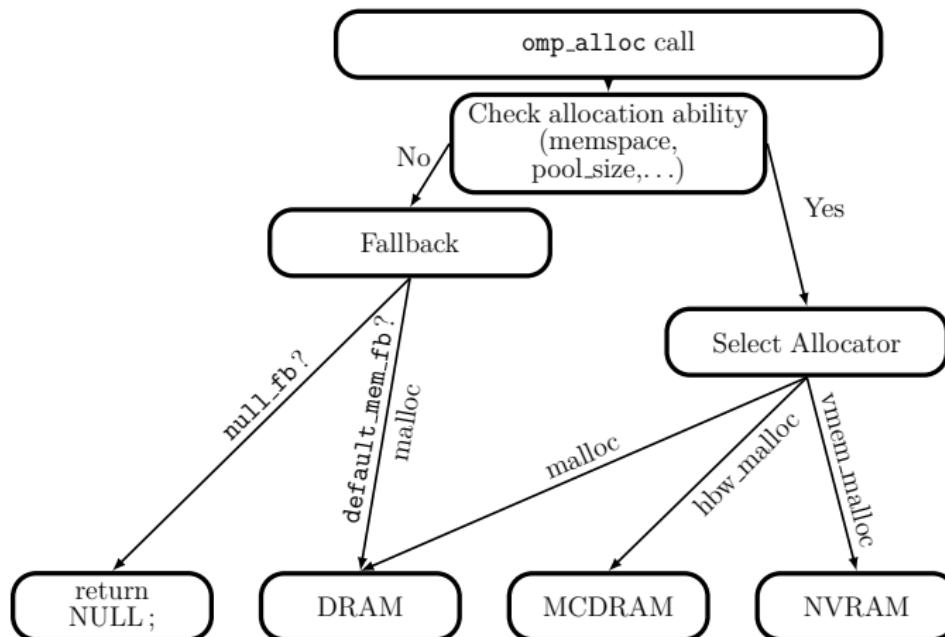












⇒ Deallocation procedure follows the same scheme

1 Introduction to OpenMP Memory Management

- Memory Management Extensions

2 Memory Management Integration into OpenMP runtime

- Hardware Detection
- Memory Allocator Initialization Process
- Data Allocation/Deallocation Process

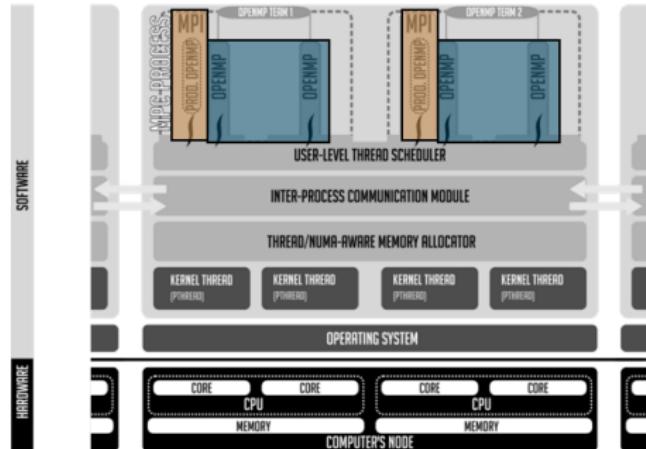
3 Implementation into the MPC framework**4** Integrating OpenMP Memory Management constructs in Software

- LULESH Application
- Memory Management Support in C++ Application

5 Evaluation**6** Conclusion and Future work

MPC is...

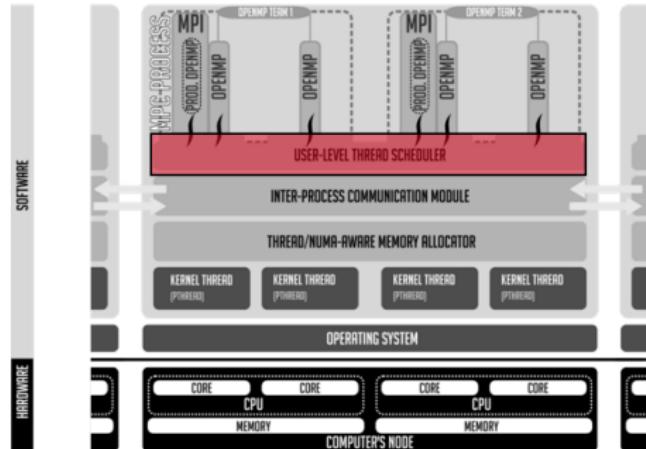
- ... a thread-based **MPI** and **OpenMP** implementations...



Open-source software
Available at <https://mpc.hpcframework.com/>

MPC is...

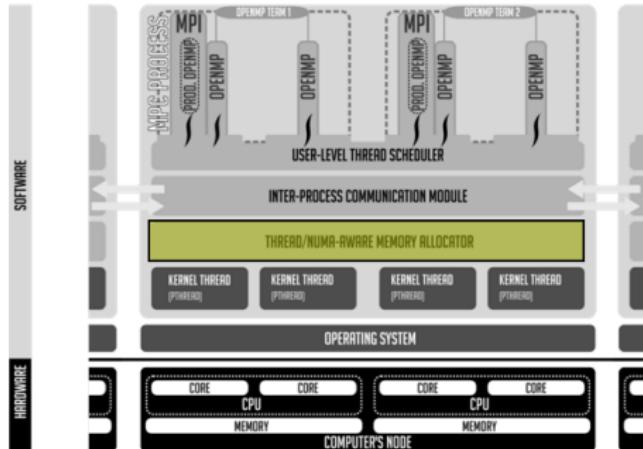
- ... a thread-based **MPI** and **OpenMP** implementations...
- ... based on the same **thread scheduler** allowing interoperability, ...



Open-source software
Available at <https://mpc.hpcframework.com/>

MPC is...

- ... a thread-based **MPI** and **OpenMP** implementations...
- ... based on the same **thread scheduler** allowing interoperability, ...
- ... and integrates a **memory allocator** avoiding NUMA effects.



Open-source software
Available at <https://mpc.hpcframework.com/>

- Support of runtime entry points only
 - MPC: runtime compatible with compilers (GCC, LLVM, Intel)

- Support of runtime entry points only
 - MPC: runtime compatible with compilers (GCC, LLVM, Intel)
- Adding Hardware discovery module for memory types detection
 - Browsing topology with hwloc 2

- Support of runtime entry points only
 - MPC: runtime compatible with compilers (GCC, LLVM, Intel)
- Adding Hardware discovery module for memory types detection
 - Browsing topology with hwloc 2
- Managing allocators in runtime system
 - Allocator identification
 - `omp_allocator_handle_t` (i.e. enum type)
 - How to manage allocators during execution ?
 - ⇒ List of allocators
 - Naive implementation
 - Global list shared among threads

- Support of runtime entry points only
 - MPC: runtime compatible with compilers (GCC, LLVM, Intel)
- Adding Hardware discovery module for memory types detection
 - Browsing topology with hwloc 2
- Managing allocators in runtime system
 - Allocator identification
 - `omp_allocator_handle_t` (i.e. enum type)
 - How to manage allocators during execution ?
 - ⇒ List of allocators
 - Naive implementation
 - Global list shared among threads
- Forward data allocation request to
 - MPC allocator for default (DRAM) and high bandwidth (MCDRAM) memories
 - VMEM library for large capacity memory (NVRAM)

- Support of runtime entry points only
 - MPC: runtime compatible with compilers (GCC, LLVM, Intel)
- Adding Hardware discovery module for memory types detection
 - Browsing topology with hwloc 2
- Managing allocators in runtime system
 - Allocator identification
 - `omp_allocator_handle_t` (i.e. enum type)
 - How to manage allocators during execution ?
 - ⇒ List of allocators
 - Naive implementation
 - Global list shared among threads
- Forward data allocation request to
 - MPC allocator for default (DRAM) and high bandwidth (MCDRAM) memories
 - VMEM library for large capacity memory (NVRAM)
- Actual supported traits
 - alignment
 - fallback

1 Introduction to OpenMP Memory Management

- Memory Management Extensions

2 Memory Management Integration into OpenMP runtime

- Hardware Detection
- Memory Allocator Initialization Process
- Data Allocation/Deallocation Process

3 Implementation into the MPC framework**4** Integrating OpenMP Memory Management constructs in Software

- LULESH Application
- Memory Management Support in C++ Application

5 Evaluation**6** Conclusion and Future work

- Presentation

- Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics
- Hydrocode
- OpenMP Implementation

- Data allocation scheme

- Dynamic memory allocations
- C++ STL vectors

- C like programs
 - Wrapping allocator calls

```
// Initialization
2  omp_memspace_handle_t mcdram = omp_high_bw_mem_space;
3  omp_allocator_traits_t mcdram_traits[1] = {omp_atk_alignment, 64};
4  omp_allocator_handle_t mcdram_alloc;
5  mcdram_alloc = omp_init_allocator(mcdram, 1, mcdram_traits);
6

// Allocation
7  void* Allocate(size_t size, omp_allocator_handle_t allocator) {
8      return (void*)omp_alloc(size, allocator);
9  }

// Deallocation
10 void Release(void **ptr, omp_allocator_handle_t allocator) {
11     if (*ptr != NULL) {
12         omp_free(*ptr, allocator) ;
13         *ptr = NULL ;
14     }
15 }
```

- C++ STL objects allocator
 - std::allocator by default
 - Add a new allocator with OpenMP Memory Management support

```
template <typename T>
2   class omp_allocator
3   {
4     public:
5       ...
6       omp_allocator_handle_t& allocator;
7
8       omp_allocator(omp_allocator_handle_t& alloc) : allocator(alloc) {}
9
10      pointer allocate(size_type n, const T* hint = 0) {
11        return (T*)omp_alloc(sizeof(T) * n, allocator);
12      }
13
14      void deallocate(pointer p, size_type n) {
15        omp_free(p, allocator);
16      }
17    };
18
19    ...
20    omp_allocator_handle_t allocator;
21    allocator = omp_init_allocator(omp_default_mem_space, 0);
22    std::vector<Real_t, omp_allocator<Real_t>> m_x(N, 0., omp_allocator<Real_t>(allocator);
```

1 Introduction to OpenMP Memory Management

- Memory Management Extensions

2 Memory Management Integration into OpenMP runtime

- Hardware Detection
- Memory Allocator Initialization Process
- Data Allocation/Deallocation Process

3 Implementation into the MPC framework**4** Integrating OpenMP Memory Management constructs in Software

- LULESH Application
- Memory Management Support in C++ Application

5 Evaluation**6** Conclusion and Future work

⇒ LULESH application

- Evaluation

- Number of elements solved per second (i.e., Figure of Merit)

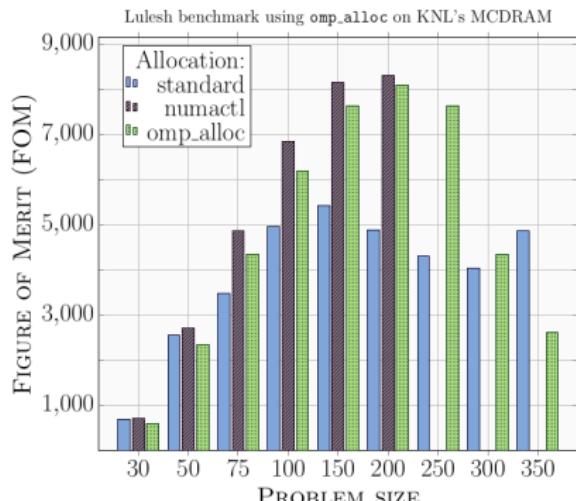
⇒ Tests performed on 2 various systems:

- Intel Knights Landing

- 68 cores @ 1.40GHz (Quadrant mode)
 - 16 GB MCDRAM (Flat mode)
 - 96 GB DDR

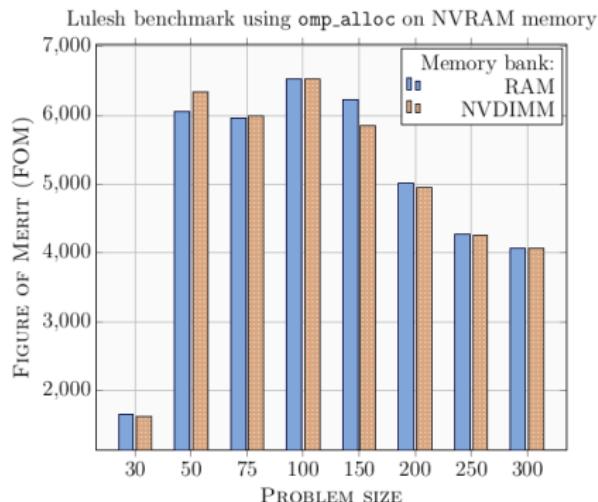
- Intel Cascade Lake

- 2 × 24 cores (Intel Xeon Platinum 8260L) @ 2.40GHz
 - 2 × NVDIMM storage (1.5 TB)



- *numactl* library
 - All allocation directed to a specific NUMA node
- Evaluation
 - no modification: with / without numactl
 - allocation through OpenMP memory management calls `omp_high_bw_mem_space`

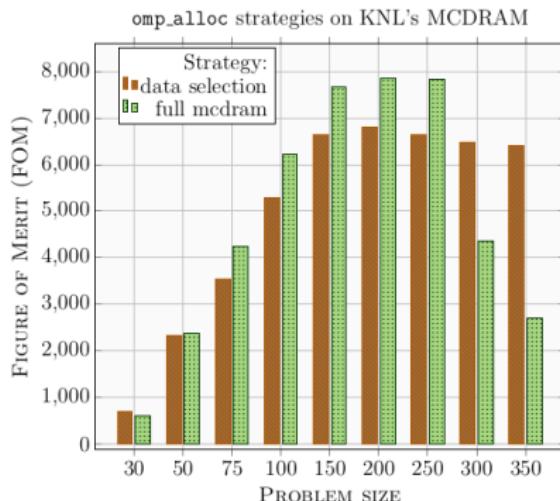
- *numactl* and OpenMP curves look similar
- *numactl* allocated a little bit more data in MCDRAM
 - MCDRAM space exceed from problem size 250



■ Allocations

- DRAM – not modified app
- NVRAM – OpenMP with `omp_large_cap_mem_space`

- No significant differences
- No error message with VMEM library
- Able to allocate memory in NVDIMM



■ Comparison

- Data selection strategy
 - Relevant data in MCDRAM
 - otherwise in DRAM
- All the data in MCDRAM

- OpenMP memory management constructs have to be used carefully

1 Introduction to OpenMP Memory Management

- Memory Management Extensions

2 Memory Management Integration into OpenMP runtime

- Hardware Detection
- Memory Allocator Initialization Process
- Data Allocation/Deallocation Process

3 Implementation into the MPC framework**4** Integrating OpenMP Memory Management constructs in Software

- LULESH Application
- Memory Management Support in C++ Application

5 Evaluation**6** Conclusion and Future work

- Preliminary implementation of OpenMP memory management constructs
 - Integration into the MPC OpenMP framework
 - Memory kinds discovery module
 - Allocator management
 - Data allocation in DRAM, MCDRAM and NVDIMM
 - Link DRAM and MCDRAM allocations with MPC allocator + NVDIMM allocations with VMEM library
- First evaluation
 - Modification of the LULESH application
 - Possible to allocate data in various memories with OpenMP
- Future Work
 - MPC release with memory management support
 - Increase the robustness of our solution
 - Supporting more trait values
 - Integrate more memory spaces

Thank you for your attention

Questions ?



Multi-**P**rocessor **C**omputing

<http://mpc.hpcframework.com/>