Hierarchical Local Storage

Exploiting Flexible User-Data Sharing Between MPI Tasks

Marc Tchiboukdjian, Patrick Carribault, Marc Pérache

Exascale computing research

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

intel

cea

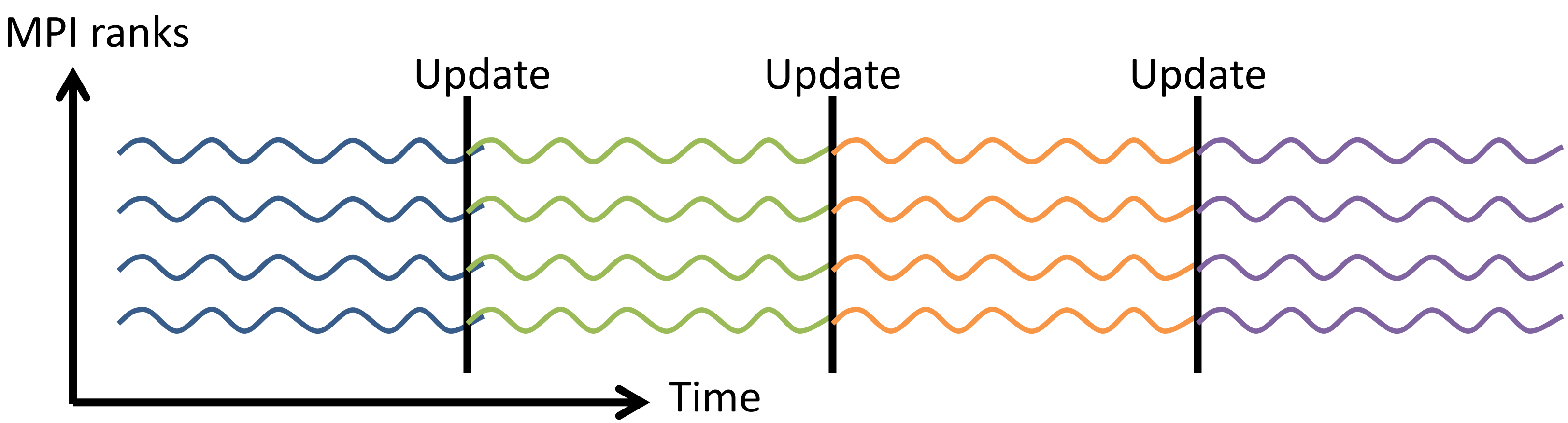
GENCI

Hierarchical Local Storage (HLS)

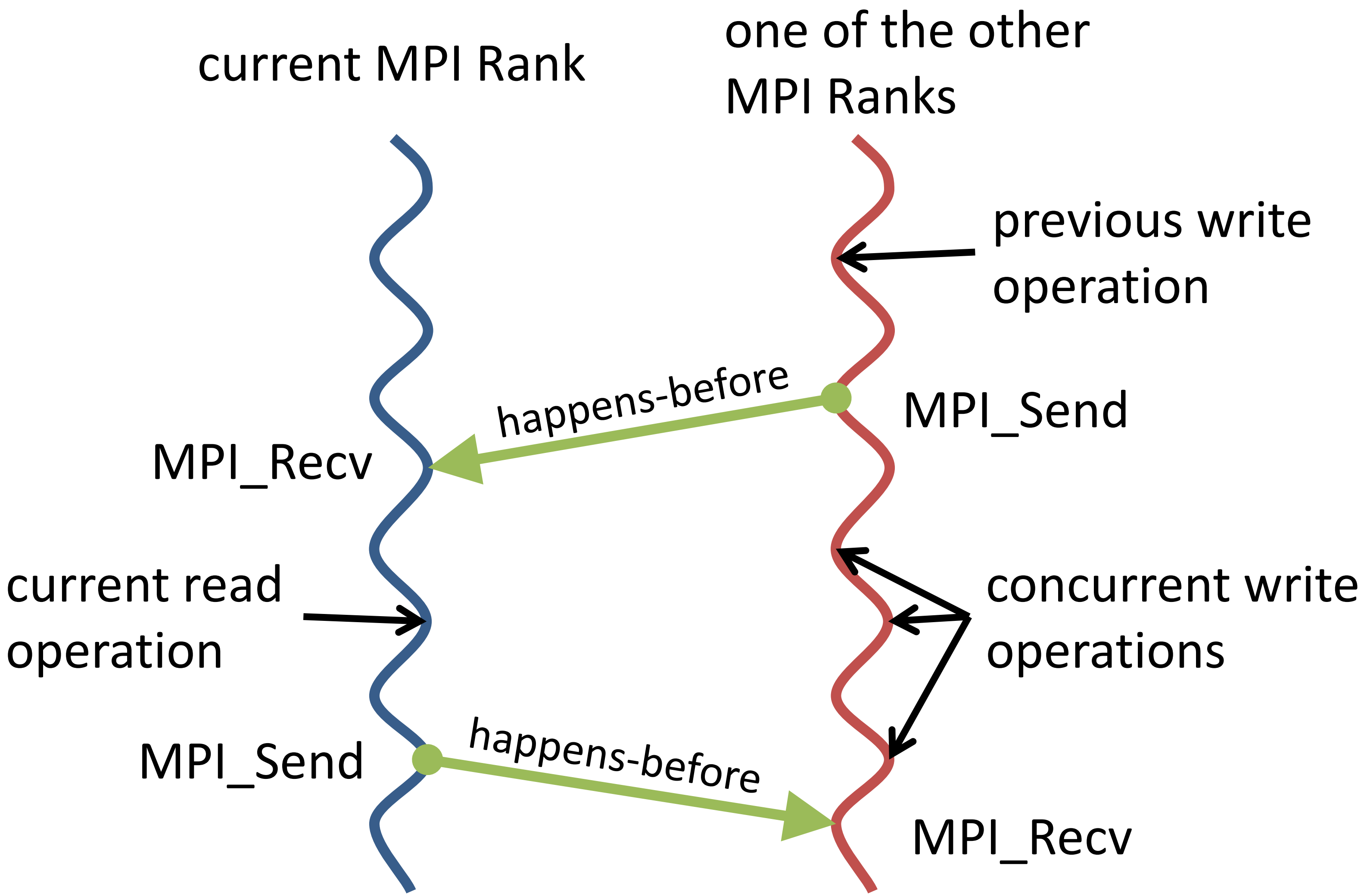
- Language extension to reduce memory consumption of MPI programs
- User can flag with pragmas data that will be shared among MPI tasks located on the same node
  - No sharing across different nodes
    $\Longrightarrow$ no additional communications
  - Take advantage of shared memory inside a node
    $\Longrightarrow$ almost no runtime overhead
- Potential memory reduction factor = #cores per node
  - From one copy per rank to one copy per node
  - HLS memory does not increase with the number of cores per node

HLS Typical Use Case: Common Variables

- Common variables
  - Same value across MPI ranks at each point of the program
  - Value can change over time but the update need to be logically synchronous for all MPI ranks
  - Examples: physics constants, replicated domain, …

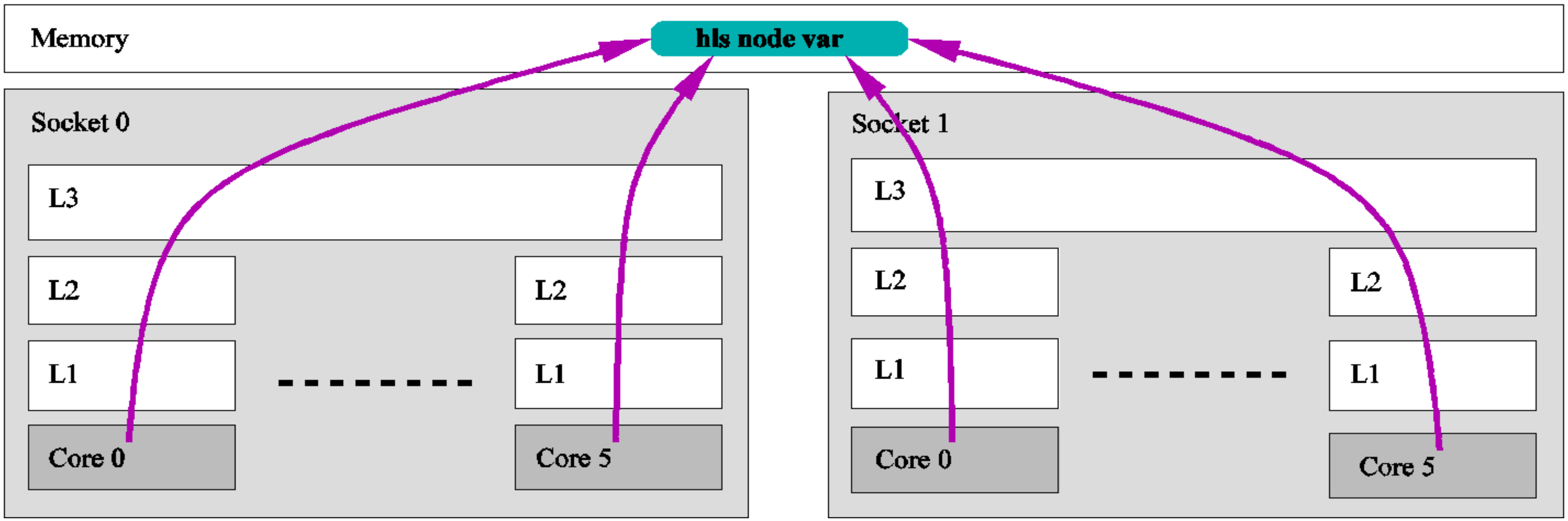


Formal Definition



Example of HLS Usage

- Example of one global variable named var
  - Duplicated in standard MPI environment
  - Shared with HLS directive
    `#pragma hls node(var)`
  - Updated with HLS directive
    `#pragma hls single(var) { ... }`

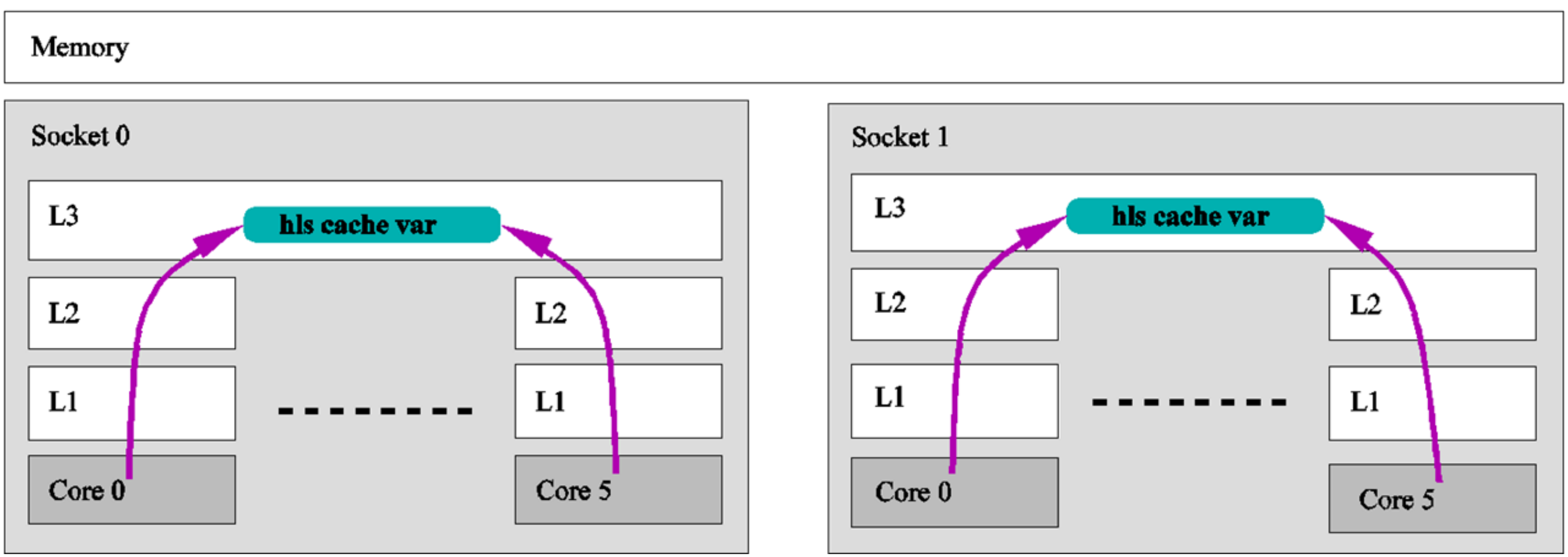


Control Data Sharing with HLS Scopes

- Sharing data across an entire node can:
  - degrade locality of MPI programs (due to NUMA effects, coherency cache misses, …)
  - induce too much synchronizations if often updated
- HLS scopes allow the user to choose at which logical level a variable should be shared
  - Available scopes: node, numa, cache level(#), core
  - Tradeoff memory consumption / runtime overhead

Example of HLS Usage with HLS Scope

- One copy per L3 cache with HLS directive
  `#pragma hls cache(var) level(3)`
- No NUMA effects, no L3 cache coherency misses, faster variable updates
- Does not guarantee that the variable will be in cache



HLS Pragmas

- #pragma hls *scope*(*list_var*) [level(*l*)]
  - variables in *list_var* are shared among all MPI ranks in the same scope
  - *scope*=node,numa,cache,...
  - *l*=1,2,3,...
  - restricted to global variables

- #pragma hls single(*list_var*) [nowait]

```
{
  .... /* modify vars in list_var */
}
```

  - code in single region is executed once per scope (e.g. once per numa node)
  - variables in *list_var* must have the same scope
  - implicit barrier before and after the single region (except with nowait)

- #pragma hls barrier(*list_var*)
  - synchronize MPI ranks at the largest scope in *list_var*

What kind of memory can be shared?

Global memory

```
double table[<big size>];
#pragma hls node(table)

void main() {
    MPI_Init();
    #pragma hls single(table)
    {
        write_table();
    }
    compute();  /* table is read only */
    MPI_Finalize();
}
```

Heap-allocated memory

```
double *table;
#pragma hls node(table)

void main() {
    MPI_Init();
    #pragma hls single(table)
    {
        table = malloc(<bigsize>*sizeof(double));
        write_table();
    }
    compute();  /* table is read only */
    MPI_Finalize();
}
```

Heap-memory allocations for HLS variables must be protected inside single regions

Implementation

- Compiler part in GCC, runtime part in MPC
- MPC: MPI 1.3 and OpenMP 2.5 runtime
  - developped at CEA and Exascale Computing Research
  - thread-based

  $\Longrightarrow$ MPI tasks on the same node share the same address space
- Similar to the Thread Local Storage (TLS) mechanism

Implementation: Compiler Part

- Patched GCC shipped with MPC release
  - Support of C, C++ and Fortran
- Parser
  - Recognize and check validity of pragmas
  - Add scope information for each variable
  - Lower barrier and single pragmas

```
#pragma hls single(a)
{
   ...
}
```

→

```
if(hls_single(node_scope)){
   ...
   hls_single_done(node_scope);
}
```

- Code generation
  - emit function calls to the MPC runtime to get the address of a variable (identified by a module and an offset)

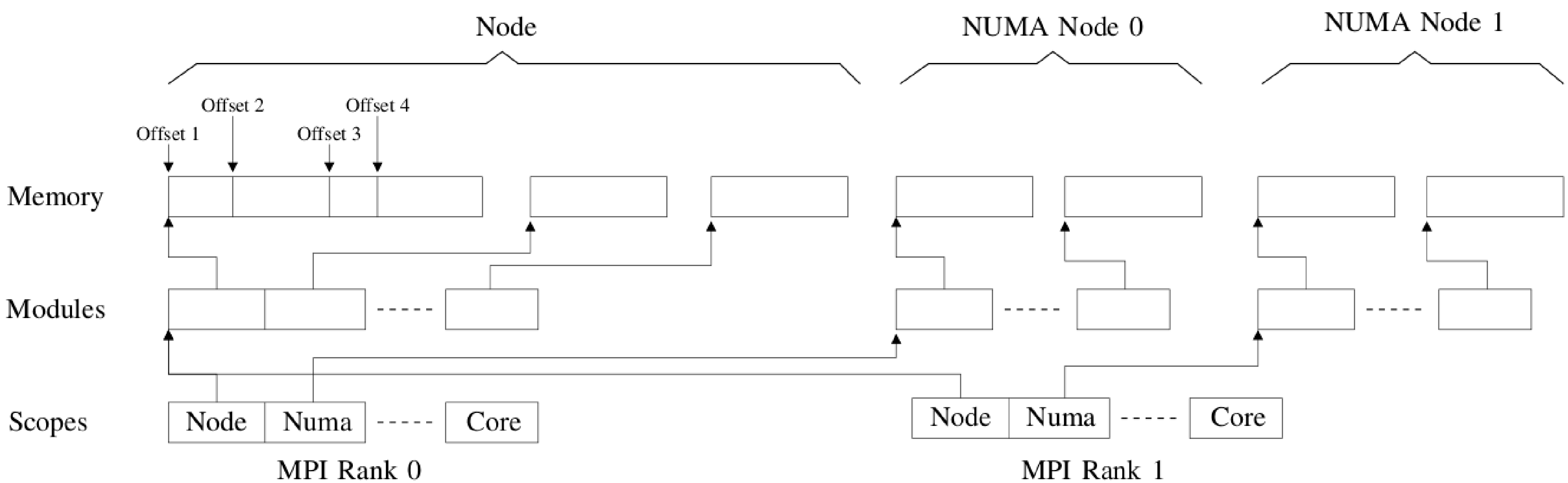
```
int a;
#pragma hls node(a)
a = 3;
```

→

```
int *ptr_a;
ptr_a = hls_get_addr_node(mod,off);
*ptr_a = 3;
```

Implementation: Runtime Part

- Using the topology, assign the correct HLS memory when creating a MPC thread

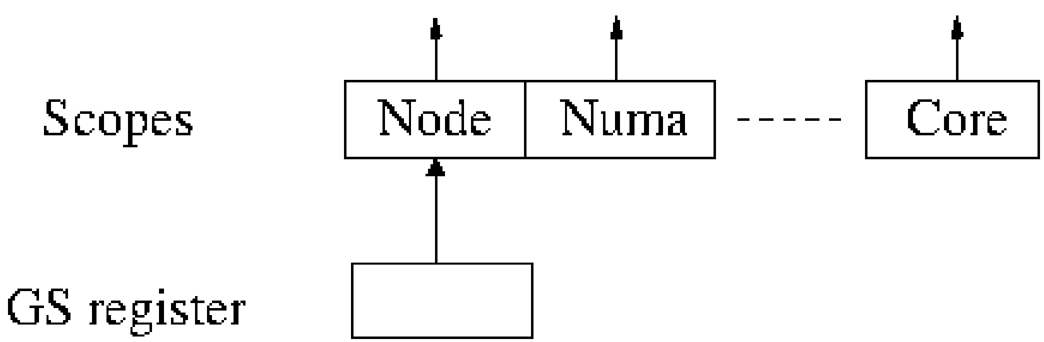


- Implement functions single, barrier and get_addr

```
void *hls_get_addr_<scope>(size_t module, size_t offset){
    // allocate and initialize memory if first use
    return hls[<scope>][mod] + off;
}
```

Implementation: Linker Part

- Impact on performances
  - At each use of an HLS variable, a function call is inserted to get its address
- This function call can be removed at link time in some cases
  - Example: linking an executable (module 0)
  - Use the segment register gs to store a pointer to the HLS scopes array (to be updated at context switch)
  - Replace the function call by some assembly code (2 instructions)

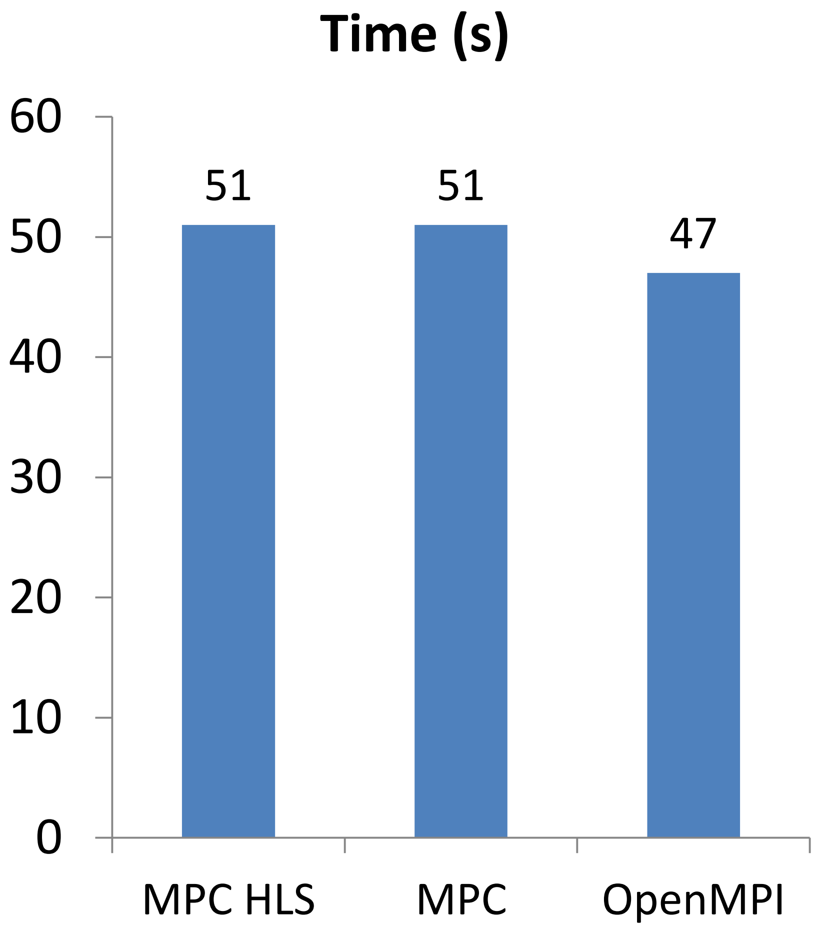


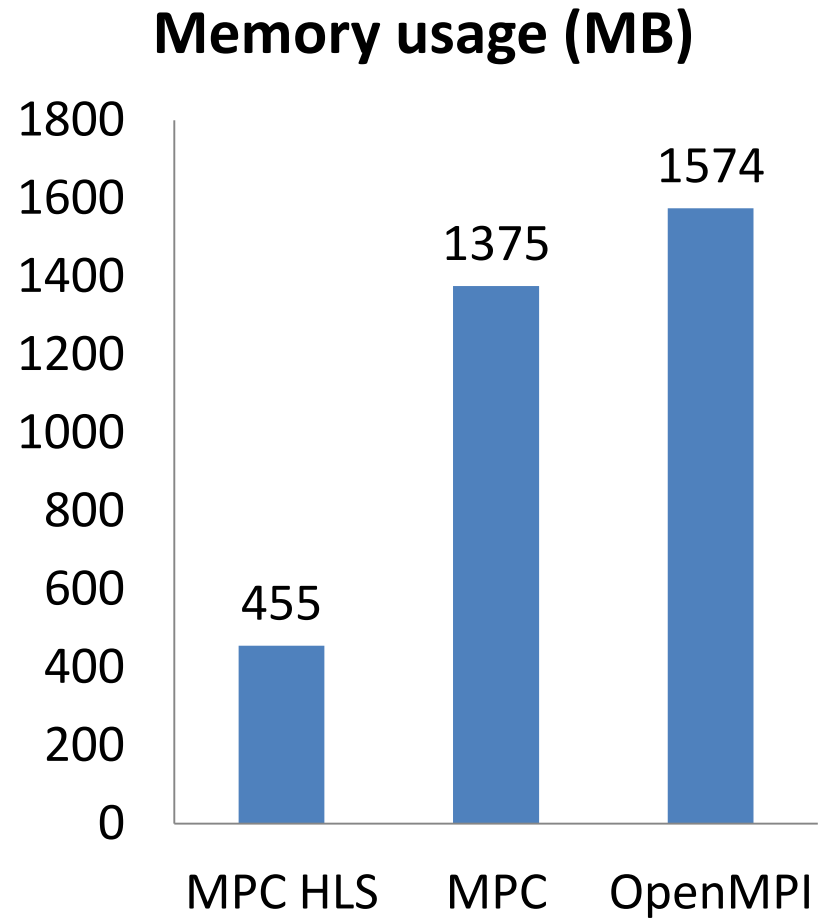
Experiments on Memory Consumption Reduction

- 3 real applications
  - EulerMHD from CEA
    HLS variable: large table storing gas behavior
  - Gadget-2 from PRACE
    HLS variable: large table storing precomputed coefficients for Ewald summation
  - Tachyon from SPEC MPI2007
    HLS variable: scene description and resulting image

- Experimental Setup
  - Comparaison between MPC 2.3.1 with and without HLS and OpenMPI 1.4.3
  - Runs on an Infiniband DDR cluster with 2-socket 4-core Core2Quad nodes

EulerMHD

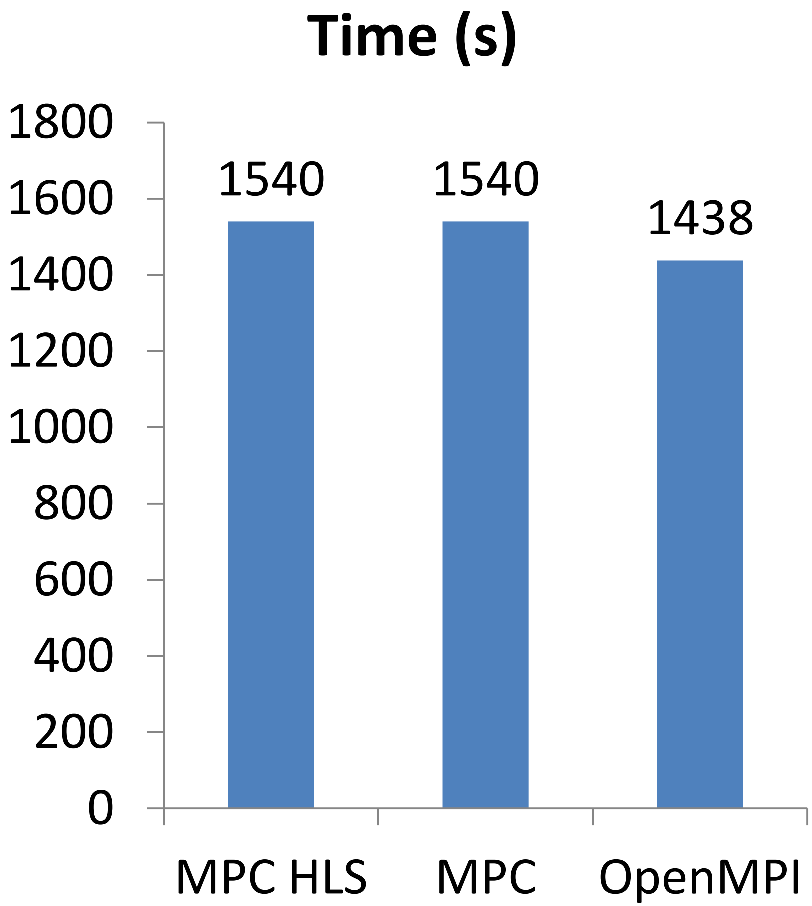


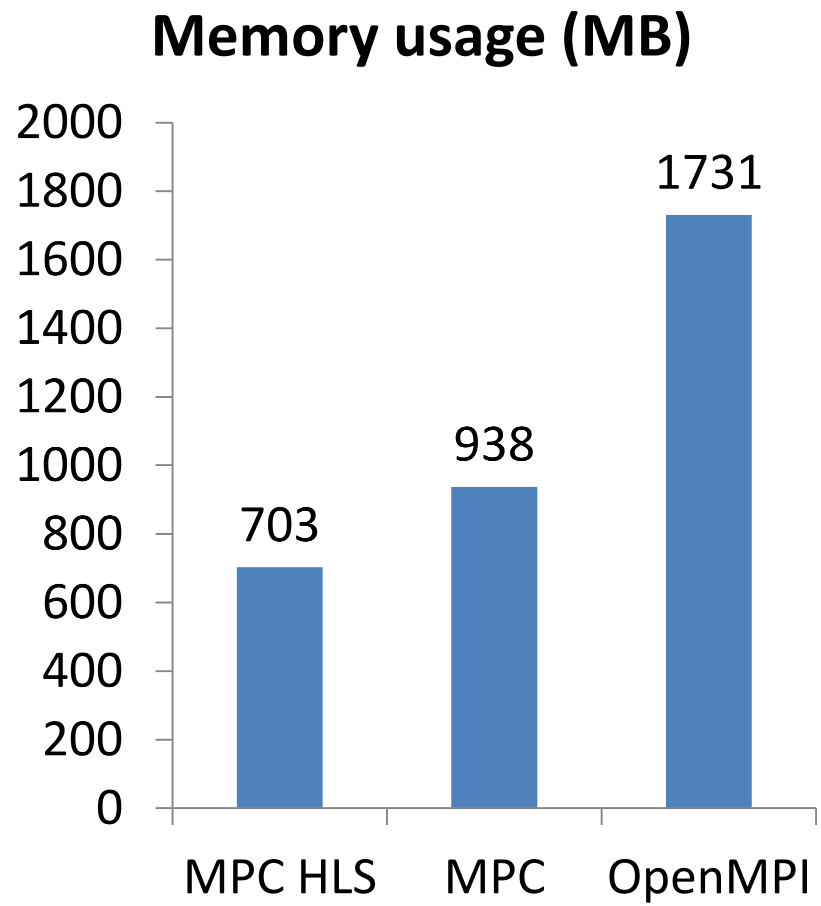


128MB table is shared by 8 cores per node

Using HLS $\Longrightarrow$ ≈900MB memory gain

Gadget-2

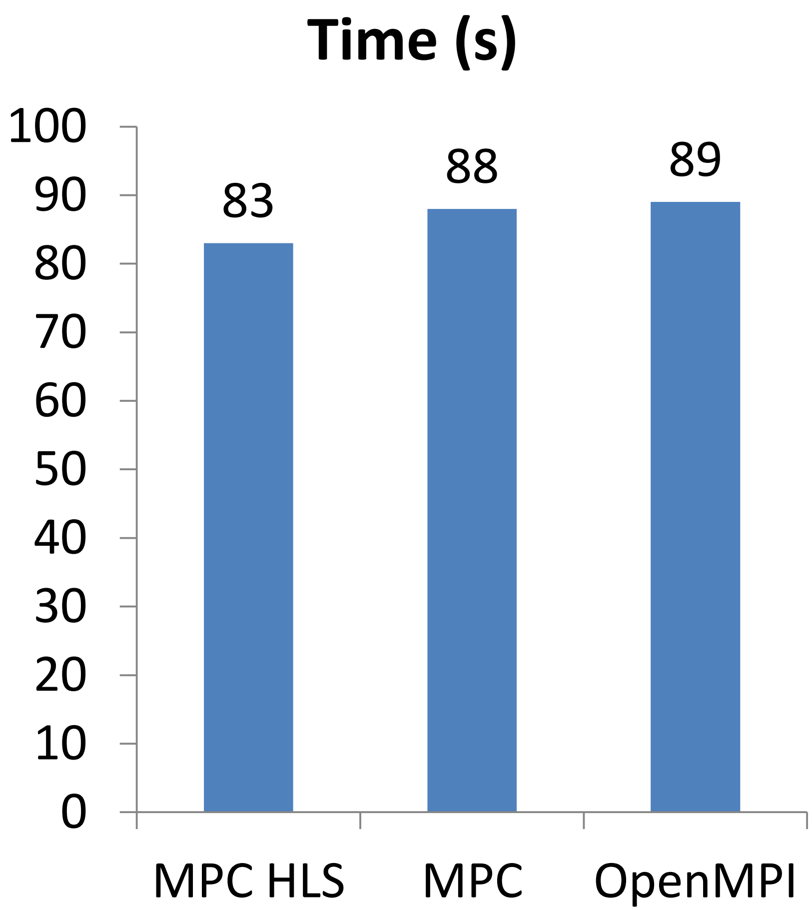


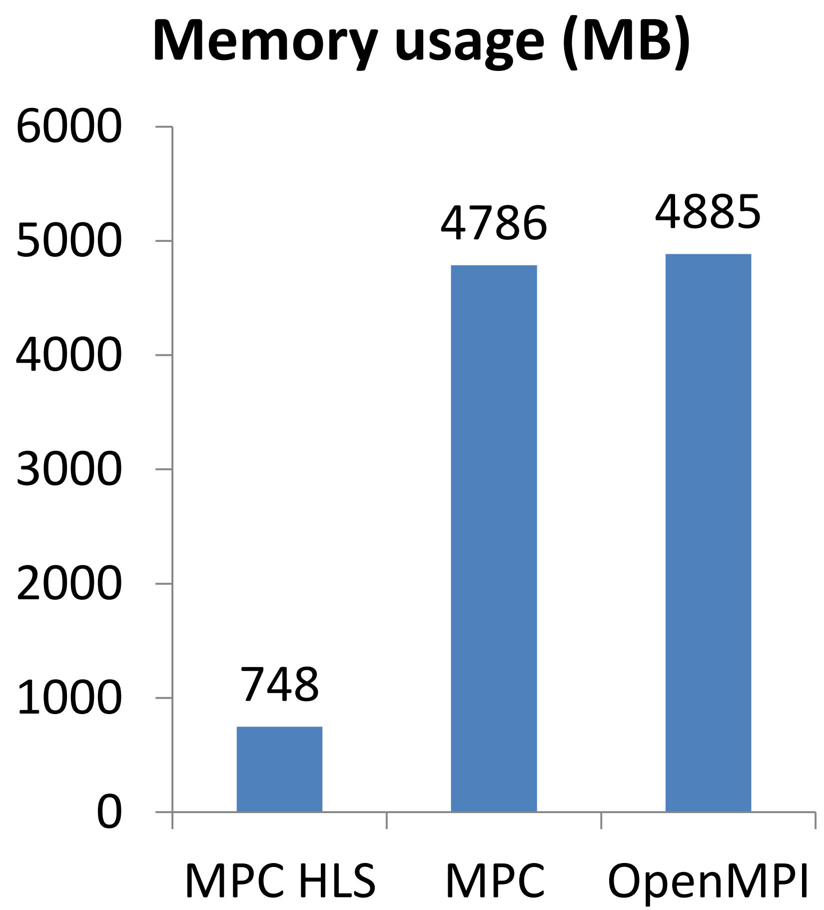


33MB table is shared by 8 cores per node

Using HLS $\Longrightarrow$ ≈230MB memory gain

Tachyon





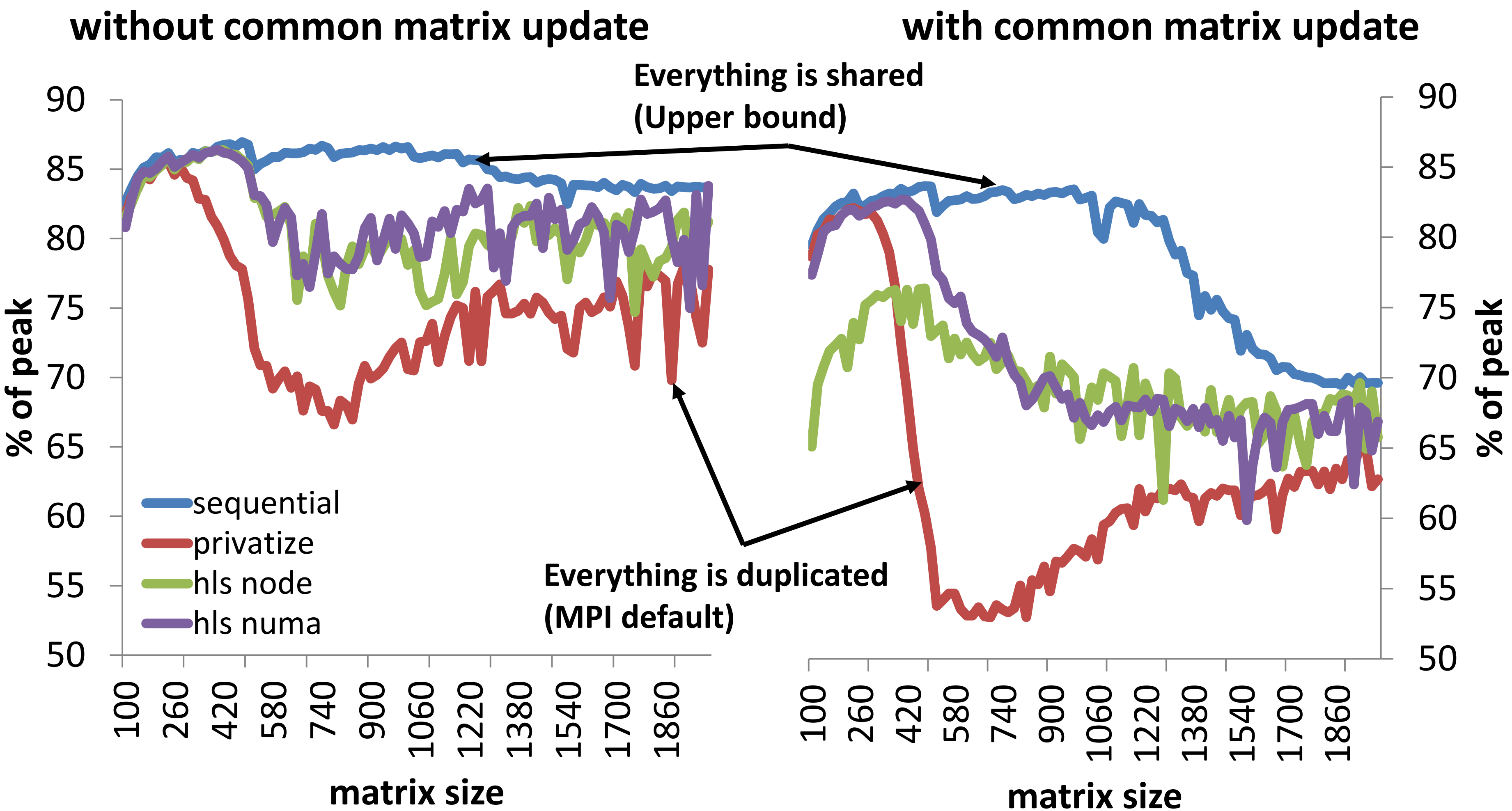
560MB scene is shared by 8 cores per node

Using HLS $\Longrightarrow$ $\approx$ 4GB memory gain

Experiments on Improved Shared Cache Usage

- 2 microbenchmarks
  - Matrix multiplication with a common matrix
  - Mesh update with common lookup table
  - These two microbenchmarks are extracted from real applications
- Experimental Setup
  - Comparaison between MPC 2.3.1 with and without HLS and a sequential run (ideal case with no data duplication)
  - Runs on a large NUMA node (4 sockets 8 cores Nehalem-EX) with a 18MB of shared cache per 8 cores
- Goal: evaluate the speedup obtained by reducing data duplication in the shared cache

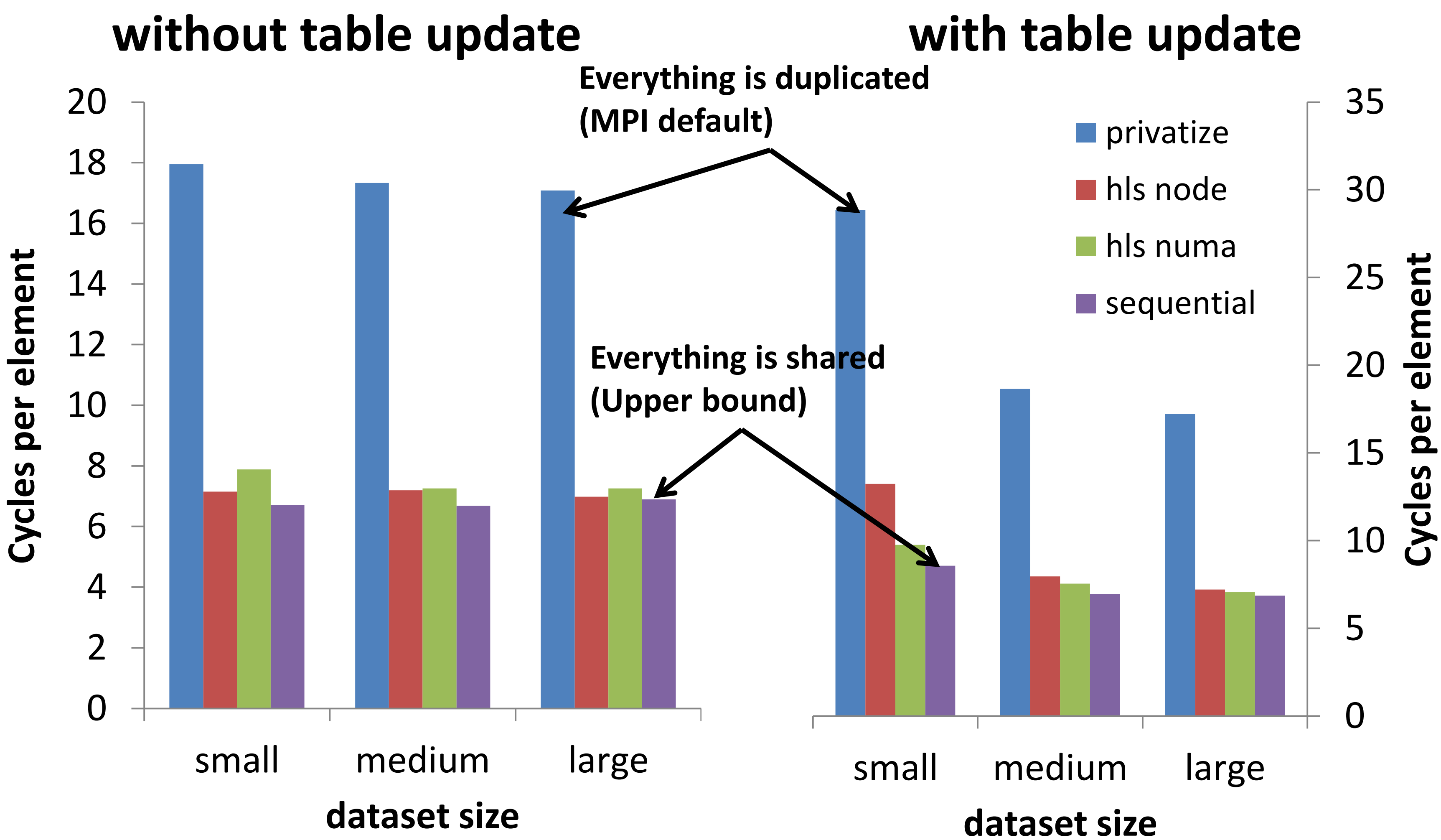
Matrix Multiplication with Common Matrix



One of the matrix is shared by 8 cores accessing the same L3 cache

Using HLS $\Longrightarrow$ up to 1.4x speedup

Mesh Update with Common Lookup Table



The table is shared by 8 cores accessing the same L3 cache

Using HLS $\Longrightarrow$ up to 3x speedup

Conclusion

- HLS is an extension to reduce memory consumption of MPI applications
  - Potential memory reduction factor = #cores per node
  - Application porting is easy on known applications
  - The patched GCC compiler and the MPC runtime are available in the MPC 2.3.1 release at http://mpc.sourceforge.net
- Currently working on a tool to automatically detect common variables

Thank you for your attention