Thread-Local Storage Extension to Support Thread-Based MPI/OpenMP Applications

Patrick Carribault, Marc Pérache and Hervé Jourdren

CEA, DAM, DIF, F-91297 Arpajon France

Introduction/Context

- HPC Architecture: Petaflop/s Era
  - Multicore processors as basic blocks
  - Clusters of ccNUMA nodes
- Parallel programming models
  - MPI: distributed-memory model
  - OpenMP: shared-memory model
- Hybrid MPI/OpenMP (or *mixed-mode programming*)
  - Promising solution (benefit from both models for data parallelism)
  - Thread-based implementation
  - How to handle data visibility?
- Contributions
  - Study on data visibility in thread-based parallel programming models
  - New mechanism to handle programming-model private data

Outline

- Introduction/Context
- Running Example
  - Distributed memory (MPI)
  - Shared memory (OpenMP)
  - Hybrid (mixed-mode) MPI/OpenMP
- Extended Thread-Local Storage (TLS)
  - Design
  - Implementation
- Experimental Results
  - Statistics on global variables
  - Overhead of extended TLS
- Conclusion & Future Work

Running Example 1/4

- Sample MPI code
  - Global variable a (catch MPI rank)
- Traditional MPI approach
  - Each MPI task is a UNIX process
  - Global variables are duplicated
- Thread-based MPI approach
  - Each MPI task is a thread
  - Process virtualization mechanism
  - Global variables are shared among MPI task located on the same node
- Need for a mechanism to duplicate global variables

```
int a = -1 ;
void f() {
  printf("Rank = %d\n", a) ;
}

int main( int argc, char ** argv ) {
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &a);
  MPI_Barrier(MPI_COMM_WORLD);
  f();
  MPI_Finalize();
}
```

Running Example 2/4

- Multiple approaches to restore data flow
- Array privatization
  - Source transformation
  - Promote variables to arrays
  - Use rank to index this array
- Thread-Local Storage (TLS)
  - Runtime mechanism
  - Need compiler cooperation
  - Local storage for every thread
  - Handle thread creation/destruction

```
int a = -1 ;
void f() {
  printf("Rank = %d\n", a) ;
}

int main( int argc, char ** argv ) {
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &a);
  MPI_Barrier(MPI_COMM_WORLD);
  f();
  MPI_Finalize();
}
```

Running Example 3/4

- Hybrid MPI/OpenMP code
  - Global variable a (catch MPI rank)
  - Used in OpenMP parallel region
- Thread-based MPI approach
  - Variable a should be duplicated for each MPI task
  - Each OpenMP thread should access the right copy of a
- Need for a mechanism to duplicate global variables and allow shared access among threads
  - Array privatization? OK
  - TLS? NO

```
int a = -1 ;

void f() {
  printf("Rank = %d\n", a ) ;
}

int main( int argc, char ** argv ) {
  MPI_Init( &argc, &argv ) ;
  MPI_Comm_rank(MPI_COMM_WORLD, &a) ;
  MPI_Barrier(MPI_COMM_WORLD);
  #pragma omp parallel
  {
    f() ;
  }
  MPI_Finalize() ;
}
```

Running Example 4/4

- Hybrid MPI/OpenMP code
  - Global variable a (catch MPI rank)
  - Variable b private to OpenMP threads
  - Both used in OpenMP parallel region
- Thread-based MPI approach
  - Variable a should be duplicated for each MPI task
  - Each OpenMP thread should access the right copy of a
  - Variable b should be duplicated for each OpenMP thread
- Contribution: extension of TLS mechanism to allow multiple levels
  - Based on TLS
  - One level for each parallel programming model

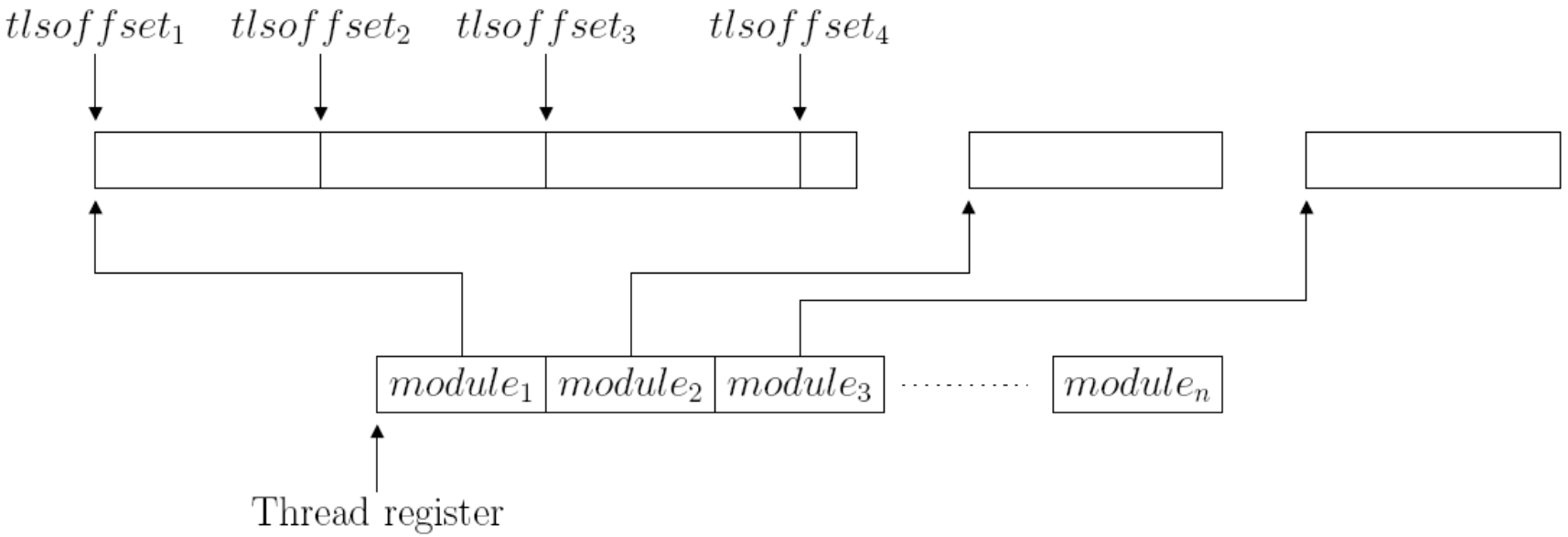
```
int a = -1 ;
int b = -1 ;
#pragma omp threadprivate(b)

void f() {
  printf( "MPI Rank = %d and"
    " OpenMP Rank = %d\n",
    a, b ) ;
}

int main( int argc, char ** argv ) {
  MPI_Init( &argc, &argv ) ;
  MPI_Comm_rank(MPI_COMM_WORLD,&a) ;
  MPI_Barrier(MPI_COMM_WORLD);
  #pragma omp parallel
  {
    b = omp_get_thread_num() ;
    f() ;
  }
  MPI_Finalize() ;
}
```

Outline

- Introduction/Context
- Running Example
  - Distributed memory (MPI)
  - Shared memory (OpenMP)
  - Hybrid (mixed-mode) MPI/OpenMP
- Extended Thread-Local Storage (TLS)
  - Design
  - Implementation
- Experimental Results
  - Statistics on global variables
  - Overhead of extended TLS
- Conclusion & Future Work

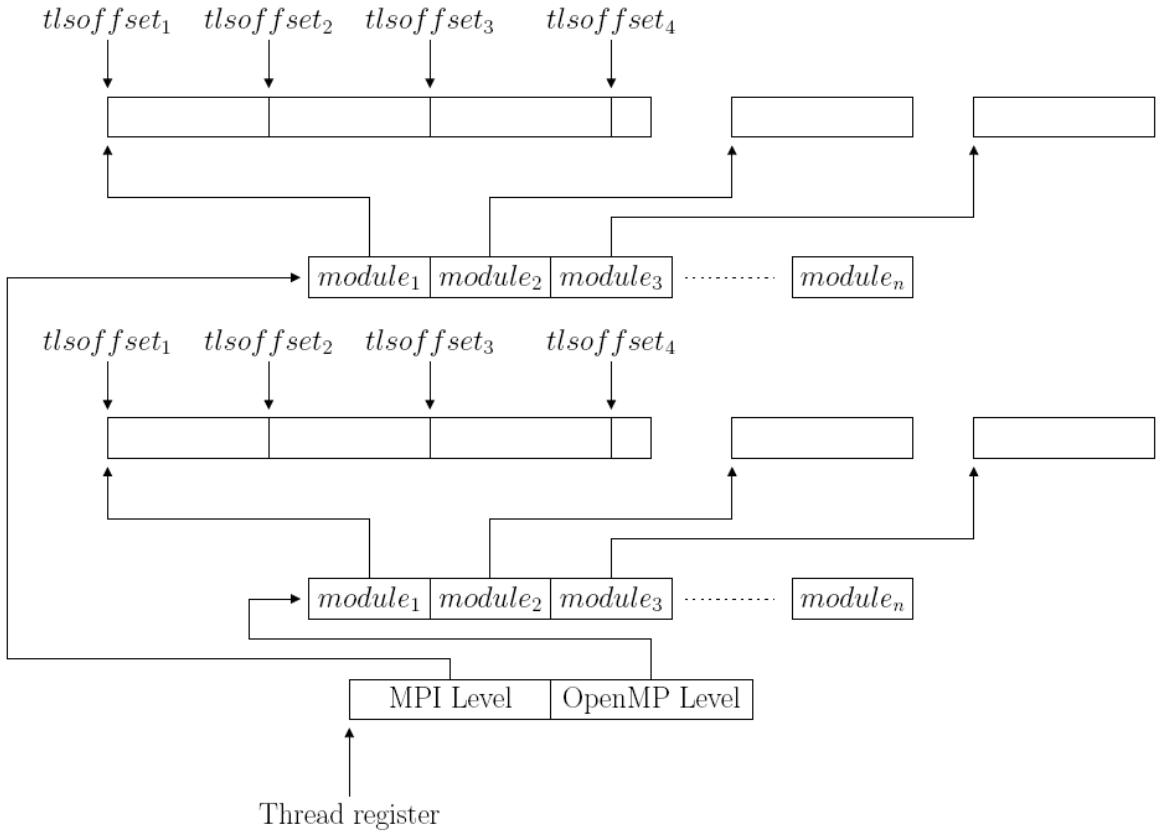
TLS Design

- Efficient and flexible usage of thread-local data
- Extension to C and C++ language
  - New keyword `__thread`
  - Cooperation between compiler/linker and runtime system
  - Full C++ support in C++0x standard (thread-local object)



Extended TLS Design

- Main idea: add the notion of levels to thread-local data
  - One additional pointer (programming-model level)



Extended TLS Implementation

- Cooperation between compiler and runtime system
  - Compiler part in GCC
  - Runtime part in MPC (Message-Passing Computing)
- Compiler part in GCC
  - New pass to put variables to the right extended-TLS level
  - Modification of backend part for code generation (link to the runtime system)
- Runtime part in MPC
  - Integrated to user-level thread mechanism
  - Copy-on-write optimization
  - Modified context switch to update pointer to extended TLS variables
- Implementation of extended TLS freely available in MPC 2.2 at `http://mpc.sourceforge.net`

Application to Parallel Programming

- Automatic privatization
  - Automatically convert any MPI code for thread-based MPI compliance
  - Put global variables in extended-TLS MPI level
- Thread-based hybrid programming
  - Automatically handle thread-based MPI/OpenMP programming model
  - Put global variables into extended-TLS MPI level
  - Put OpenMP `threadprivate` variables into OpenMP level
- Embedding external tools
  - Automatically convert external tools (JITs, interpreters, VMs …)
  - Possibility to embed these tools in threads

Outline

- Introduction/Context
- Running Example
  - Distributed memory (MPI)
  - Shared memory (OpenMP)
  - Hybrid (mixed-mode) MPI/OpenMP
- Extended Thread-Local Storage (TLS)
  - Design
  - Implementation
- Experimental Results
  - Statistics on global variables
  - Overhead of extended TLS
- Conclusion & Future Work

Experimental Results

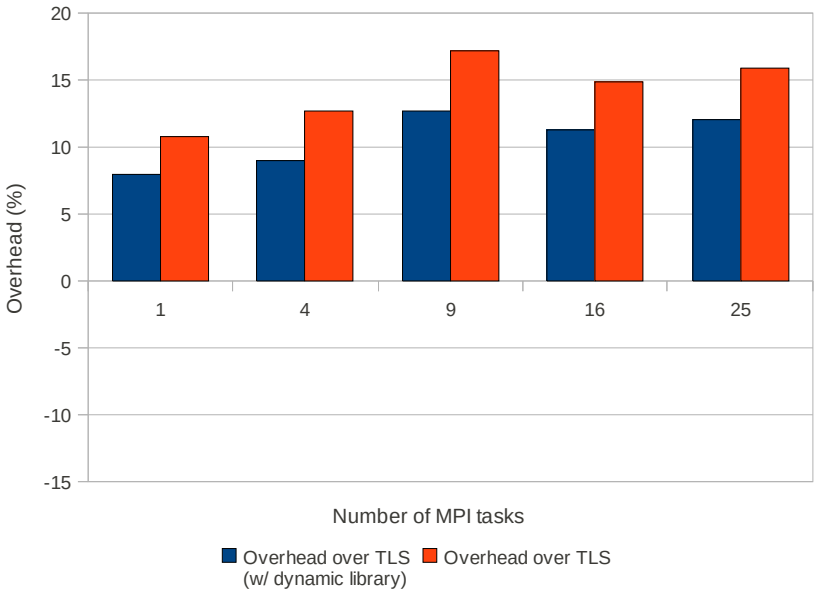
- Evaluation of extended TLS overhead compared to regular TLS
- Target architecture
  - TERA 100 machine
  - As of November 2010, rank 6th in Top500
  - Quad-socket eight-core Nehalem EX nodes
- Statistics
  - Number of variables to update per benchmark
- Experimental results
  - NAS benchmark
  - MPI, OpenMP and MultiZone version

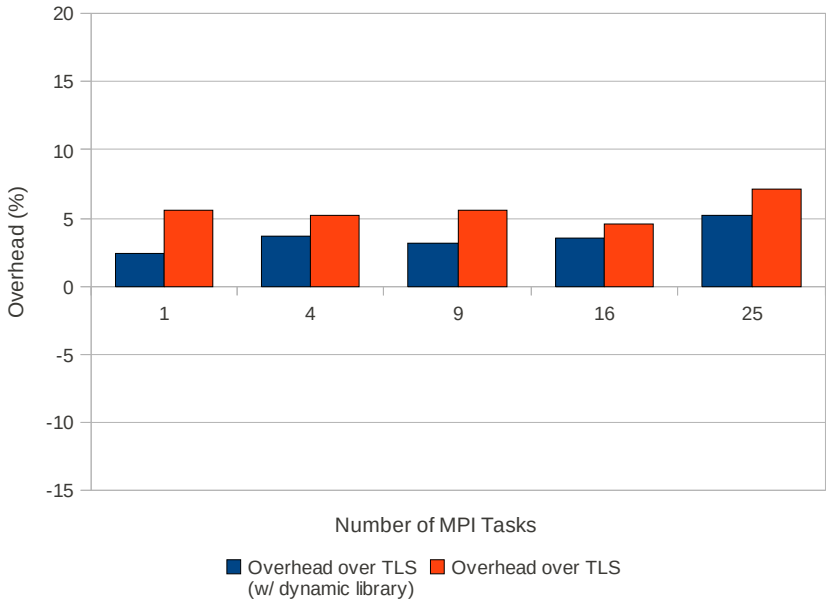
Experimental Results

- NAS MultiZone benchmarks
  - High-level language: Fortran
  - Parallel programming models: MPI+OpenMP
- Statistics on the number of variables to update for thread-safety compliance

Statistics	BT	LU	SP
Language	Fortran	Fortran	Fortran
Lines of code	5,154	4,618	5,085
Number of global variables	173	175	132
Accesses to global variables	258	274	273
Number of threadprivate variables	11	11	5
Accesses to threadprivate variables	44	50	58

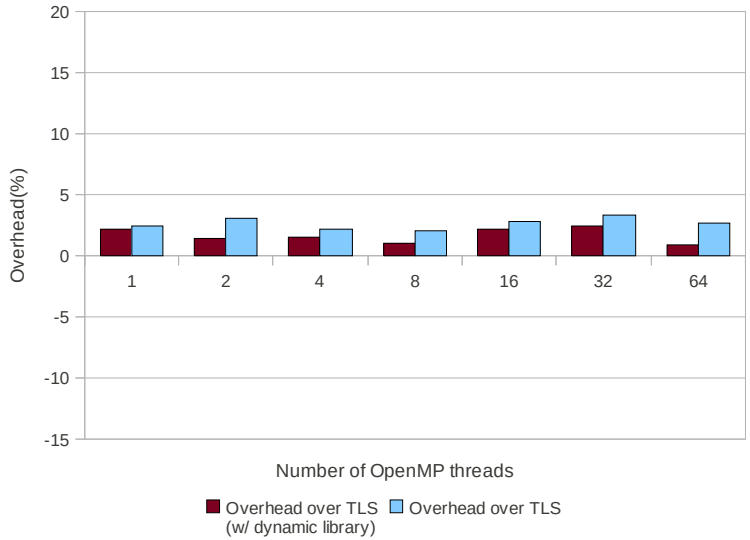
Experimental Results

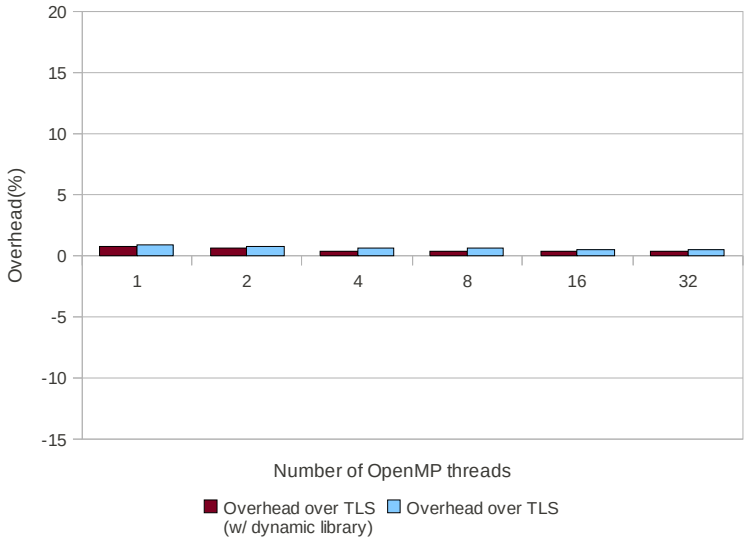




- Experiments on NAS 3.3 MPI class B (left: SP, right: BT)
- Overhead of extended TLS approach compared to regular TLS
  - Overhead up to 15% (depending on benchmark)
  - Dynamic library hampers TLS optimizations (large part of overhead)
  - Extended TLS hampers linker optimizations

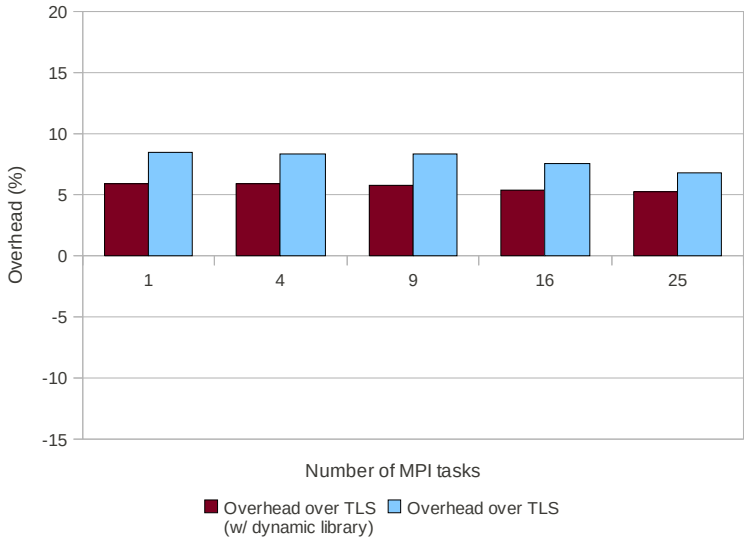
Experimental Results

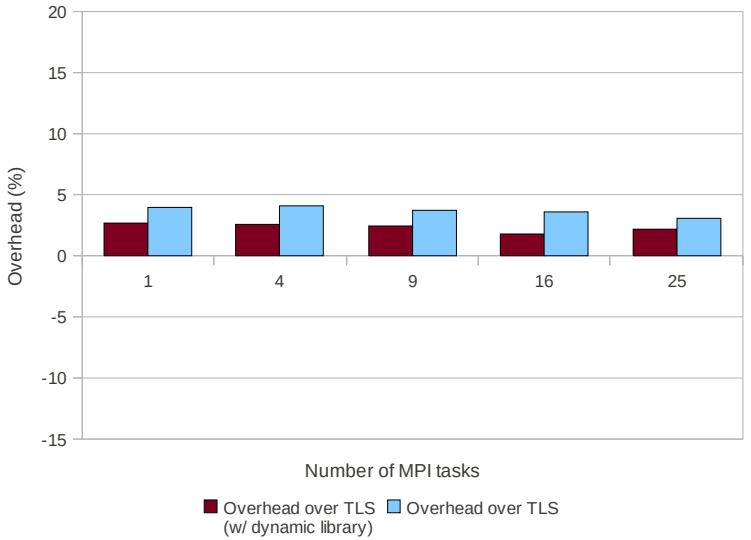




- Experiments on NAS 3.3 OpenMP class B (left: SP, right: BT)
- Overhead of extended TLS approach compared to executable with TLS
  - Threadprivate variables are put into TLS
  - Smaller overhead

Experimental Results





- Experiments on NAS MZ 3.2 class B (left: SP, right: BT)
- Overhead of extended TLS approach compared to executable with TLS
  - Global variables and threadprivate variables are put into extended TLS
  - Overhead up to 10% (mainly because of no TLS optimizations)

Outline

- Introduction/Context
- Running Example
  - Distributed memory (MPI)
  - Shared memory (OpenMP)
  - Hybrid (mixed-mode) MPI/OpenMP
- Extended Thread-Local Storage (TLS)
  - Design
  - Implementation
- Experimental Results
  - Statistics on global variables
  - Overhead of extended TLS
- Conclusion & Future Work

Conclusion

- Context
  - Mixing multiple thread-based parallel programming models
- Contributions
  - New mechanism based on extension of TLS (Thread-Local Storage)
  - Cooperation between compiler (GCC) and runtime system (MPC)
  - Freely available at http://mpc.sourceforge.net (version 2.2)
    - Contact: patrick.carribault@cea.fr or marc.perache@cea.fr
- Future Work
  - Optimization of linker tool to hoist TLS accesses
  - Deal with more applications/embedded tools
  - Extend this mechanism to more than 2 programming models