



Recent developments in Archer: New analyses and support for OpenMP 6.0

Joachim Jenke, 16th IPTW 2025



Why do we need data race detection?

C++

- The execution of a program contains a **data race** if it contains **two** potentially **concurrent** conflicting actions, at least one of which is **not atomic**, and **neither happens before** the other, [...]. Any such data race results in **undefined behavior**.

OpenMP

- **Multiple threads** access the same memory **unordered**, at least one thread writes. If a **data race** occurs then the result of the program is **unspecified**.
- There is **no benign** data race in C/C++/OpenMP. It is always **UB!**

Undefined Behavior: What could go wrong?

```
int test(int n, char *c) {  
    if (n == 0)  
        return 0;  
    return atoi(c);  
}
```

```
int main(int argc, char **argv) {  
    char *buf;  
    if (argc > 1) {  
        buf = strdup(argv[1]);  
    }  
    int res = test(argc - 1, buf);  
    return res;  
}
```

Similar MPI example:

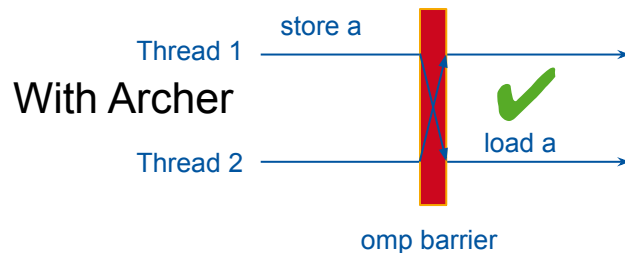
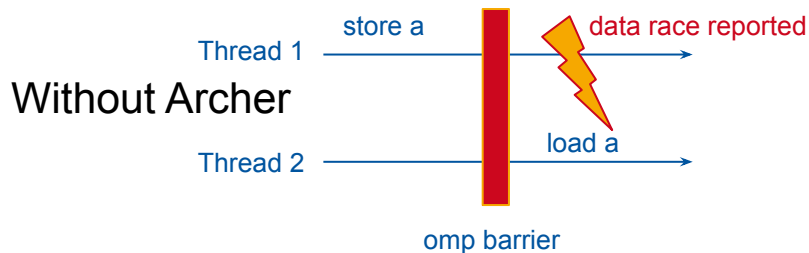
```
float *lbuf, *rbuf;  
lbuf = malloc(n * sizeof(float));  
if (rank == 0) {  
    rbuf = malloc(n * sizeof(float));  
}  
MPI_Reduce(lbuf, rbuf, n, MPI_FLOAT, ...);
```

```
test:  
    test    edi, edi  
    je     .LBB0_1  
    push   rax  
    mov    rdi, rsi  
    xor    esi, esi  
    mov    edx, 10  
    call   strtol  
    add    rsp, 8  
    ret  
  
.LBB0_1:  
    xor    eax, eax  
    ret  
  
main:  
    push   rax  
    mov    rdi, qword ptr [rsi + 8]  
    call   strdup  
    mov    rdi, rax  
    xor    esi, esi  
    mov    edx, 10  
    pop    rax  
    jmp    strtol
```

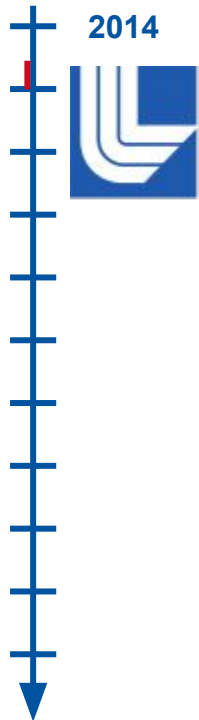
Unconditional
call to strdup

Why do we need Archer?

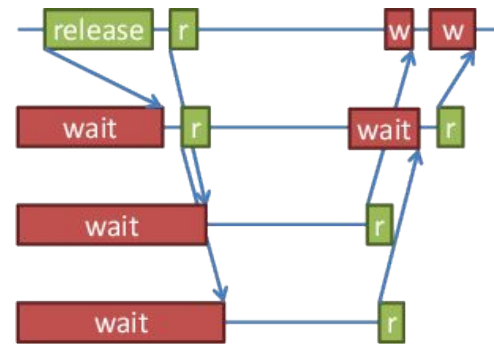
- valgrind and ThreadSanitizer are agnostic of OpenMP synchronization
- To use valgrind/TSan with libgomp: build with `--disable-linux-futex` 🤔
- Archer understands all OpenMP synchronization (and concurrency) semantics and feeds them into TSan's happens-before analysis



First implementation



- Internship at LLNL 6/14 - 9/14
- Initial implementation using TSan annotations in libomp
 - OFF by default, configure with LIBOMP_TSAN_SUPPORT
 - Complementary compiler pass to whitelist race-free pieces of code
 - Presented as paper at 1st LLVM@SC 2014 workshop
- Code available from separate github repo



Upstreamed to LLVM



2014

2015

2016

- May '16: IPDPS paper about static analysis
- October '16: libomp patch pushed to llvm repository
 - Available with LLVM 4.0 release
 - Disabled in default build, enabled in Spack package
- June '16: First commit for the OMPT-based tool



Finally in LLVM



2014

2015

2016

2017

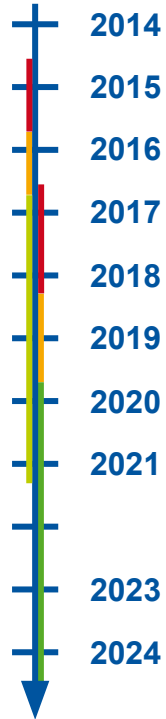
2018

2019

- March '19: Updates to make the tool compatible with OMPT 5.0
- November '19: OMPT-based Archer landed
 - Archer is active by default now (if application is built with TSan)
 - Available since LLVM 10 release



Today

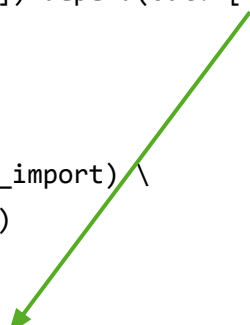


- Intel shifted towards sanitizers
 - Inspector was discontinued
 - Archer is available and on by default with Intel compilers (2024.2)
- HPE/Cray compiler supports use of Archer
- AMD's AOCC compiler packages include Archer
- Several patches (e.g., AVX512 for TSan) pending in review

What's New From OpenMP 6.0?

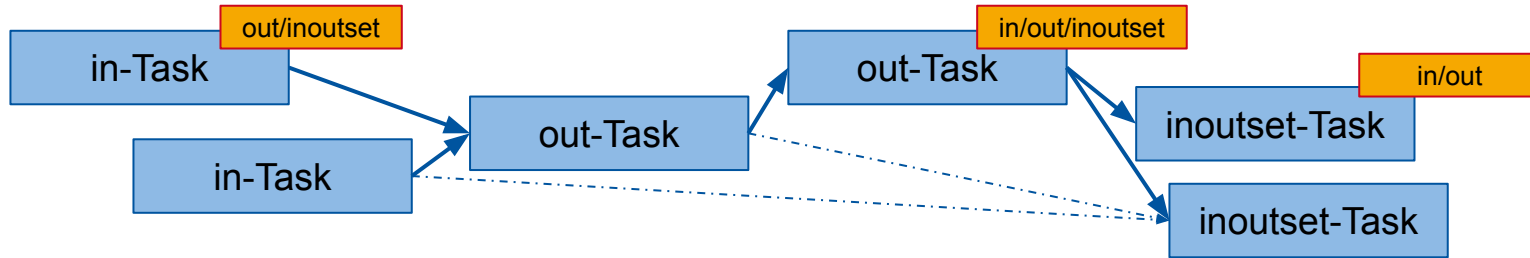
OpenMP 6.0: Transparent tasks

```
#pragma omp task transparent(omp_export) \  
    depend(out:dummy)  
{  
    for(int i=0; i<MAX_TASKS; i++)  
        #pragma omp task depend(in:A[i]) depend(out:B[i])  
            foo(A[i], B[i]);  
}  
  
#pragma omp task transparent(omp_import) \  
    depend(in:dummy)  
{  
    for(int i=0; i<MAX_TASKS; i++)  
        #pragma omp task depend(in:B[i]) depend(out:C[i])  
            bar(B[i], C[i]);  
}
```



- In general, OpenMP task dependencies only apply to siblings
- Transparent tasks import or export dependencies of their child tasks

- `hashmap<void*, DepObject*>` per task, created on demand
 - Access only on creation of child tasks → no locking necessary
 - Only read, insert, delete. No remove.
- Tasks with dependencies store references to the `DepObject*` along with the dependence kind
- `DepObject` maintains Synchronization Clocks (SC) for `in`, `out`, and `inoutset`.
- dependence kind determines which SC is used to acquire synchronization.
Release synchronization only to dependent kinds



Transparent Task Dependencies in Archer

- Imported and exported dependencies reside in the parent task's hashmap
- Transparent tasks can be nested
 - Nesting only has an effect for nested import or export chains
- Local dependencies are still possible
- The transparent task might need to add new entries to the hashmap → possibly concurrent access to the hashmap → hashmap needs locking
- For import/export, only acquiring/releasing dependencies wrt. parent task

Implementation impact

- Code for tracking dependencies: 15 LoC $\rightarrow$ 80 LoC (5x)
- Code for annotating dependencies: 6 LoC $\rightarrow$ 18 LoC (3x)
- In total 140 new LoC: 1276 $\rightarrow$ 1416 LoC (+11%)

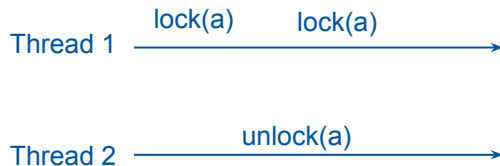
- Locking of hashmap access necessary, as soon as one child task is transparent

- Performance impact:
 - No implementation of transparent tasks is available
 - Experiment with Fibonacci tasking code shows no performance impact

New Features Beyond Data Race Detection

Detect Misuse of OpenMP Locks

- A lock-releasing routine must not access a lock that is owned by a task other than the encountering task.
- Effectively the restriction boils down to “the thread that acquired the lock must release the lock”.
 - Such TSan analysis is already available for pthread mutex
- Application use case: misuse as conditional variable



Allow turning Analysis off/on

- Archer already allows to ignore serial region code
 - (ARCHER_OPTIONS=ignore_serial=1)
- Using the `omp_tool_control` function, an application can modify tool behavior
 - OpenMP defines values for start, pause, flush and end
- We use `pause+start` to disable+enable TSan analysis

Summary & Outlook

- With endurance, tenacity, and some effort a summer project can become an industry standard tool
- Archer is the tool to integrate and test new OMPT features

Next Step:

- Transfer changes into OTF-CPT

