(OpenMP) Parallelism Aware Optimizations

Johannes, Joseph, Stefan, Giorgis, Hamilton, Shilei

OpenMP in LLVM

Weekly Meeting: https://bit.ly/2Zqt49v

Johannes Doerfert
johannesdoerfert@gmail.com
Argonne National Lab

OpenMP in LLVM

Weekly Meeting: https://bit.ly/2Zqt49v

Clang

OpenMP Parser

OpenMP Sema

OpenMP CodeGen

OpenMP in LLVM

Weekly Meeting: https://bit.ly/2Zqt49v

Clang

OpenMP Parser

OpenMP Sema

OpenMP CodeGen

OpenMP runtimes

libomp.so (classic, host)

libomptarget + plugins (offloading, host)

libomptarget-nvptx (offloading, device)

OpenMP in LLVM

Weekly Meeting: https://bit.ly/2Zqt49v

Flang

Clang

OpenMP Parser

OpenMP Sema

OpenMP CodeGen

OpenMP runtimes

libomp.so (classic, host)

libomptarget + plugins (offloading, host)

libomptarget-nvptx (offloading, device)

OpenMP in LLVM

Weekly Meeting: https://bit.ly/2Zqt49v

Flang

Clang

OpenMP Parser

OpenMP Sema

OpenMP CodeGen

OpenMPIRBuilder

frontend-independant OpenMP LLVM-IR generation

favor simple and expressive LLVM-IR

reusable for non-OpenMP parallelism

OpenMP runtimes

libomp.so (classic, host)

libomptarget + plugins (offloading, host)

libomptarget-nvptx (offloading, device)

OpenMP in LLVM

Weekly Meeting: https://bit.ly/2Zqt49v

Flang

Clang

OpenMP Parser

OpenMP Sema

OpenMP CodeGen

OpenMPIRBuilder

frontend-independant OpenMP LLVM-IR generation

favor simple and expressive LLVM-IR

reusable for non-OpenMP parallelism

OpenMPOpt

interprocedural optimization pass

contains host & device optimizations

run with -O2 and -O3 since LLVM 11

OpenMP runtimes

libomp.so (classic, host)

libomptarget + plugins (offloading, host)

libomptarget-nvptx (offloading, device)

OpenMP in LLVM

Weekly Meeting: https://bit.ly/2Zqt49v

Flang

Clang

OpenMP Parser

OpenMP Sema

OpenMP CodeGen

OpenMPIRBuilder

frontend-independant OpenMP LLVM-IR generation

favor simple and expressive LLVM-IR

reusable for non-OpenMP parallelism

OpenMPOpt

interprocedural optimization pass

contains host & device optimizations

run with `-O2` and `-O3` since LLVM 11

OpenMP runtimes

libomp.so (classic, host)

libomptarget + plugins (offloading, host)

libomptarget-nvptx (offloading, device)

Joseph Huber
huberjn@ornl.gov
Oak Ridge National Lab

Design Goal

Report every successful and failed optimization

Optimization Remarks

Example: OpenMP runtime call deduplication

```
double *A = malloc(size * omp_get_thread_limit());
double *B = malloc(size * omp_get_thread_limit());

#pragma omp parallel
do_work(&A[omp_get_thread_num()*size]);
#pragma omp parallel
do_work(&B[omp_get_thread_num()*size]);
```

OpenMP runtime calls with same return values can be merged to a single call

Optimization Remarks

Example: OpenMP runtime call deduplication

```c
double *A = malloc(size * omp_get_thread_limit());
double *B = malloc(size * omp_get_thread_limit());

#pragma omp parallel
do_work(&A[omp_get_thread_num()*size]);
#pragma omp parallel
do_work(&B[omp_get_thread_num()*size]);
```

OpenMP runtime calls with same return values can be merged to a single call

```
$ clang -g -O2 deduplicate.c -fopenmp -Rpass=openmp-opt
```

```
deduplicate.c:12:29: remark: OpenMP runtime call omp_get_thread_limit moved to deduplicate.c:11:29: [-Rpass=openmp-opt]
    double *B = malloc(size*omp_get_thread_limit());
deduplicate.c:11:29: remark: OpenMP runtime call omp_get_thread_limit deduplicated [-Rpass=openmp-opt]
    double *A = malloc(size*omp_get_thread_limit());
```

Design Goal

Communicate and explain OpenMP implementation details to users

Advisor Remarks

Example: OpenMP Target Scheduling

```c
#pragma omp target teams distribute parallel for collapse(2)
for (int i = 0; i < M; ++i)
    for (int j = 0; j < N; ++j)
      body(i, j);
```

```c
#pragma omp target teams distribute
for (int i = 0; i < M; ++i) {
#pragma omp parallel for
    for (int j = 0; j < N; ++j)
      body(i, j);
}
```

Advisor Remarks

Example: OpenMP Target Scheduling

```
#pragma omp target teams distribute parallel for collapse(2)
for (int i = 0; i < M; ++i)
    for (int j = 0; j < N; ++j)
      body(i, j);
```

Good Performance

```
#pragma omp target teams distribute
for (int i = 0; i < M; ++i) {
#pragma omp parallel for
    for (int j = 0; j < N; ++j)
      body(i, j);
}
```

Bad* Performance

** First optimization to provide better performance in this case already available, don't prematurely optimize your code!*

Advisor Remarks

Example: OpenMP Target Scheduling

```
#pragma omp target teams distribute parallel for collapse(2)
for (int i = 0; i < M; ++i)
    for (int j = 0; j < N; ++j)
      body(i, j); // SPMD Mode
```

SPMD mode evenly distributes work among the blocks and threads

```
#pragma omp target teams distribute
for (int i = 0; i < M; ++i) {
#pragma omp parallel for
    for (int j = 0; j < N; ++j)
      body(i, j); // Generic Mode
}
```

Generic mode requires a complex state machine to schedule the threads

Advisor Remarks

Example: OpenMP Target Scheduling

```
clang -Rpass=openmp-opt ...
```

```
void bar(void) {
   #pragma omp parallel
   { }
}
void foo(void) {
  #pragma omp target teams
  {
   #pragma omp parallel
   {}
   bar();
   #pragma omp parallel
   {}
  }
}
```

remark: Found a parallel region that is called in a target region but not part of a combined target construct nor nested inside a target construct without intermediate code. This can lead to excessive register usage for unrelated target regions in the same translation unit due to spurious call edges assumed by ptxas.
remark: Parallel region is not known to be called from a unique single target region, maybe the surrounding function has external linkage?; will not attempt to rewrite the state machine use.
remark: Found a parallel region that is called in a target region but not part of a combined target construct nor nested inside a target construct without intermediate code. This can lead to excessive register usage for unrelated target regions in the same translation unit due to spurious call edges assumed by ptxas.
remark: Specialize parallel region that is only reached from a single target region to avoid spurious call edges and excessive register usage in other target regions. (parallel region ID: __omp_outlined__1_wrapper, kernel ID: __omp_offloading_35_a1e179_foo_l7)
remark: Target region containing the parallel region that is specialized. (parallel region ID: __omp_outlined__1_wrapper, kernel ID: __omp_offloading_35_a1e179_foo_l7)
remark: Found a parallel region that is called in a target region but not part of a combined target construct nor nested inside a target construct without intermediate code. This can lead to excessive register usage for unrelated target regions in the same translation unit due to spurious call edges assumed by ptxas.
remark: Specialize parallel region that is only reached from a single target region to avoid spurious call edges and excessive register usage in other target regions. (parallel region ID: __omp_outlined__3_wrapper, kernel ID: __omp_offloading_35_a1e179_foo_l7)
remark: Target region containing the parallel region that is specialized. (parallel region ID: __omp_outlined__3_wrapper, kernel ID: __omp_offloading_35_a1e179_foo_l7)
remark: OpenMP GPU kernel __omp_offloading_35_a1e179_foo_l7

Advisor Remarks and Runtime

- Communicating OpenMP implementation details can get complicated
- Maintain a webpage with extra information and implementation details
  - https://openmp.llvm.org/<information_id>
- Add support for additional information from the runtime library

```
$ clang -O2 generic.c -fopenmp -fopenmp-targets=nvptx64-nvidia-cuda -o generic
$ env LIBOMPTARGET_INFO=1 ./generic
```

```
CUDA device 0 info: Device supports up to 65536 CUDA blocks and 1024 threads with a warp size of 32
CUDA device 0 info: Launching kernel __omp_offloading_fd02_c2a59832_main_l106 with 48 blocks and 128 threads in Generic mode
```

Stefan Stipanovic
stefomeister@gmail.com
University of Novi Sad

Google Summer of Code

Design Goal

Allow modular OpenMP code without performance penalty

no need for manual low-level optimizations

Tracking OpenMP Internal Control Variables

```
void apply(void (*func)(), int N) {
  if (omp_in_parallel()) {


  } else {



  }
}
```

Tracking OpenMP Internal Control Variables

```
void apply(void (*func)(), int N) {
  if (omp_in_parallel()) {
    for (int i = 0; i < N; ++i)
       func(i);
  } else {
    #pragma omp parallel for
    for (int i = 0; i < N; ++i)
       func(i);
  }
}
```

Tracking OpenMP Internal Control Variables

```
void apply(void (*func)(), int N) {
  if (omp_in_parallel()) {
    for (int i = 0; i < N; ++i)
       func(i);
  } else {
    #pragma omp parallel for
    for (int i = 0; i < N; ++i)
       func(i);
  }
}
```

Can be deleted if `omp_in_parallel()` is known to return true.

Tracking OpenMP Internal Control Variables

ICV Tracking allows us to:

- Replace runtime calls with known values
- Use known values for other optimizations
- Done interprocedurally through Attributor integration
- Not limited to ICVs defined by the OpenMP standard, e.g., track if in a spmd target region (implementation defined state).

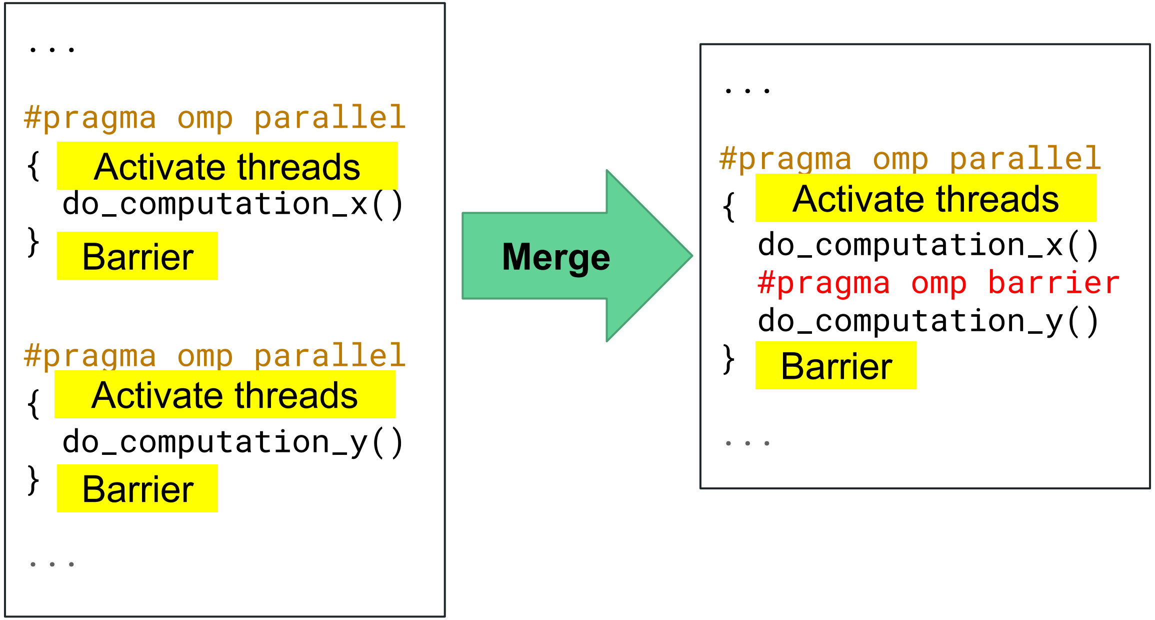
Giorgis Georgakoudis
georgakoudis1@llnl.gov
Lawrence Livermore N. Lab

Design Goal

Allow modular OpenMP code without performance penalty

no need for manual high-level optimizations

Parallel Region Merging Optimization



Parallel Region Merging Optimization

```
...

#pragma omp parallel
{  Activate threads
  do_computation_x()
}  Barrier


do_sequential_work()

#pragma omp parallel
{  Activate threads
  do_computation_y()
}  Barrier

...
```

Merge

```
...

#pragma omp parallel
{  Activate threads
  do_computation_x()
  #pragma omp barrier

  #pragma omp master {
    do_sequential_work()
  }
  #pragma omp barrier

  do_computation_y()
}  Barrier

...
```

Only if unsafe to run in parallel

Google Summer of Code

Hamilton Tobon Mosquera
hamiltontobon77@gmail.com
EAFIT University

Design Goal

Allow modular OpenMP code without performance penalty

no need for manual high-level optimizations

Hide Latency from Host to Device Memory Transfers

- Try to hide the latency of runtime calls that involve a host to device memory transfer.
- Split these memory transfers into a non-blocking “issue” of the transfer and a “wait” until the transfer is completed.
- The “issue” is moved upwards in the function until finding an instruction that may modify one of the memory regions transferred.
- The “wait” is moved downwards with the same principle, but also stopping at other runtime calls that require that memory regions in the device.
- Hopefully, when another runtime call requires the memory it will already be in the device.

Hide Latency from Host to Device Memory Transfers

```
void process_array(double * restrict a, unsigned size) {

    some_computation();

    #pragma omp target data map(a[0:size], size)
    #pragma omp target teams
    for (int i = 0; i < size; i++)
        compute(a[i]);
}
```

Hide Latency from Host to Device Memory Transfers

```
void process_array(double * restrict a, unsigned size) {

    #pragma omp target data map(a[0:size], size) depend(out:transfer) nowait
    some_computation(); // We ensure this computation does not modify *a nor size.

    #pragma omp taskwait depend(in:transfer)
    #pragma omp target data map(a[0:size], size)
    #pragma omp target teams // *a and size hopefully in device's memory already.
    for (int i = 0; i < size; i++)
        compute(a[i]);
}
```

Hide Latency from Host to Device Memory Transfers

```
void process_array(double * restrict a, unsigned size) {

    handle_t h = issue_data_map(a, size);
    some_computation(); // We ensure this computation does not modify *a nor size.

    wait_data_map(h);
    #pragma omp target data map(a[0:size], size)
    #pragma omp target teams // *a and size hopefully in device's memory already.
    for (int i = 0; i < size; i++)
        compute(a[i]);
}
```

Hide Latency from Host to Device Memory Transfers

```
void process_array(double * restrict a, unsigned size) {
    #pragma omp target data map(tofrom: a[0:size], size)
    #pragma omp target teams
    for (int i = 0; i < size; i++)
        first_transformation(a[i]);


    some_computation();


    #pragma omp target data map(tofrom: a[0:size], size)
    #pragma omp target teams
    for (int i = 0; i < size; i++)
        second_transformation(a[i]);
}
```

Hide Latency from Host to Device Memory Transfers

```
void process_array(double * restrict a, unsigned size) {
    #pragma omp target data map(to: a[0:size], size)
    #pragma omp target teams
    for (int i = 0; i < size; i++)
        first_transformation(a[i]);
    handle_t h1 = issue_data_map_back(a, size);

    some_computation(); // we make sure this does not use *a nor size

    wait_data_map(h1);
    #pragma omp target data map(tofrom: a[0:size], size)
    #pragma omp target teams
    for (int i = 0; i < size; i++)
        second_transformation(a[i]);
}
```

Hide Latency from Host to Device Memory Transfers

```
void process_array(double * restrict a, unsigned size) {
    #pragma omp target data map(to: a[0:size], size)
    #pragma omp target teams
    for (int i = 0; i < size; i++)
        first_transformation(a[i]);


    some_computation(); // we make sure this does not modify *a nor size.


    #pragma omp target data map(from: a[0:size], size) // no need to send *a nor size to the device.
    #pragma omp target teams
    for (int i = 0; i < size; i++)
        second_transformation(a[i]);
}
```

Shilei Tian
shilei.tian@stonybrook.edu
Stony Brook University

Design Goal

Optimize offloading code

perform host + accelerator optimizations

Heterogeneous LLVM-IR Module

user_code_1.c

```
void foo() {
  int N = 1024;

#pragma omp target
  *mem = N;
}
```

* RFC: https://bit.ly/3bJzAx3

Heterogeneous LLVM-IR Module

user_code_1.c

```c
void foo() {
  int N = 1024;

#pragma omp target
  *mem = N;
}
```

host.c

```c
extern void device_func7(int);

void foo() {
  int N = 1024;

  if (!offload(device_func7, N)) {
    // host fallback
    *mem = N;
  }
}
```

device.c

```c
void device_func7(int N) {
  *mem = N;
}
```

* RFC: https://bit.ly/3bJzAx3

Heterogeneous LLVM-IR Module

user_code_1.c

```
void foo() {
  int N = 1024;

#pragma omp target
  *mem = N;
}
```

host.c

```
extern void device_func7(int);

void foo() {
  int N = 1024;

  if (!offload(device_func7, 1024)) {
    // host fallback
    *mem = 1024;
  }
}
```

device.c

```
void device_func7(int N) {
  *mem = N;
}
```

* RFC: https://bit.ly/3bJzAx3

Heterogeneous LLVM-IR Module

user_code_1.c

```
void foo() {
  int N = 1024;

#pragma omp target
  *mem = N;
}
```

host.c

```
extern void device

void foo() {
  int N = 1024;

  if (!offload(device_func7, 1024)) {
    // host fallback
    *mem = 1024;
  }
}
```

The constant is part of the "host code".

device.c

```
void device_func7(int N) {
  *mem = N;
}
```

* RFC: https://bit.ly/3bJzAx3

Heterogeneous LLVM-IR Module

user_code_1.c

```c
void foo() {
  int N = 1024;

#pragma omp target
  *mem = N;
}
```

heterogeneous.c

```c
__attribute__((callback(Func, ...)))
int offload(void (*)(...) Func, ...);

target 0 void foo() {
  int N = 1024;

  if (!offload(device_func7, N)) {
    // host fallback
    *mem = N;
  }
}

target 1 void device_func7(int N) {
  *mem = N;
}
```

* RFC: https://bit.ly/3bJzAx3

* callback attribute: https://bit.ly/32I68ds

Heterogeneous LLVM-IR Module

user_code_1.c

```c
void foo() {
  int N = 1024;

#pragma omp target
  *mem = N;
}
```

heterogeneous.c

```c
__attribute__((callback(Func, ...)))
int offload(void (*)(...) Func, ...);

target 0 void foo() {
  int N = 1024;

  if (!offload(device_func7, N)) {
    // host fallback
    *mem = 1024;
  }
}

target 1 void device_func7(int N) {
  *mem = 1024;
}
```

* RFC: https://bit.ly/3bJzAx3

* callback attribute: https://bit.ly/32l68ds

Heterogeneous LLVM-IR Module

user_code_2.c

```c
void foo() {
  int a[8], b[8];

#pragma omp target
  for (int i = 0; i < 8; ++i)
    a[i] = b[i];
}
```

* RFC: https://bit.ly/3bJzAx3

Heterogeneous LLVM-IR Module

user_code_2.c

```
void foo() {
  int a[8], b[8];

#pragma omp target
  for (int i = 0; i < 8; ++i)
    a[i] = b[i];
}
```

host.c

```
extern void device_func7(int*, int*);

void foo() {
  int a[8], b[8];

  if (!offload(device_func7, a, b)) {
    // host fallback
    for (int i = 0; i < 8; ++i)
      a[i] = b[i];
  }
}
```

device.c

```
void device_func7(int *a, int *b) {
  for (int i = 0; i < 8; ++i)
      a[i] = b[i];
}
```

* RFC: https://bit.ly/3bJzAx3

Heterogeneous LLVM-IR Module

user_code_2.c

```c
void foo() {
  int a[8], b[8];

#pragma omp target
  for (int i = 0; i < 8; ++i)
    a[i] = b[i];
}
```

host.c

```c
extern void device_func7(int*, int*);

void foo() {
  int a[8], b[8];

  if (!offload(device_func7, a, b)) {
    // host fallback
    for (int i = 0; i < 8; ++i)
      a[i] = b[i];
  }
}
```

Map Types	
a	tofrom
b	tofrom

device.c

```c
void device_func7(int *a, int *b) {
  for (int i = 0; i < 8; ++i)
      a[i] = b[i];
}
```

* RFC: https://bit.ly/3bJzAx3

Heterogeneous LLVM-IR Module

user_code_2.c

```c
void foo() {
  int a[8], b[8];

#pragma omp target
  for (int i = 0; i < 8; ++i)
    a[i] = b[i];
}
```

host.c

```c
extern void device_func7(int*, int*);

void foo() {
  int a[8], b[8];

  if (!offload(device_func7, a, b)) {
    // host fallback
    for (int i = 0; i < 8; ++i)
      a[i] = b[i];
  }
}
```

Map Types	
a	tofrom
b	tofrom

device.c

```c
void device_func7(int *a, int *b) {
  for (int i = 0; i < 8; ++i)
      a[i] = b[i];
}
```

* RFC: https://bit.ly/3bJzAx3

Heterogeneous LLVM-IR Module

user_code_2.c

```
void foo() {
  int a[8], b[8];

#pragma omp target
  for (int i = 0; i < 8; ++i)
    a[i] = b[i];
}
```

heterogeneous.c

```
__attribute__((callback(Func, ...)))
int offload(void (*)(...) Func, ...);

target 0 void foo() {
  int a[8], b[8];

  if (!offload(device_func7, a, b)) {
    // host fallback
    for (int i = 0; i < 8; ++i)
      a[i] = b[i];
  }
}

target 1
void device_func7(int *a, int *b) {
  for (int i = 0; i < 8; ++i)
      a[i] = b[i];
}
```

Map Types	
a	from
b	to

* RFC: https://bit.ly/3bJzAx3

Johannes Doerfert
johannesdoerfert@gmail.com
Argonne National Lab

(Future) Design Goal

Expand beyond OpenMP by generalizing the functionality

~~Recap~~ Future Work

- ~~OpenMP~~ runtime call deduplication
- Infrastructure for improved ~~OpenMP-specific~~ feedback (remarks and more)
- Interprocedural tracking of (hidden) ~~OpenMP~~ (runtime) state
- ~~OpenMP~~ parallelism aware optimizations
- ~~OpenMP~~ target memory transfer optimizations
- ~~OpenMP~~ host-device optimizations

Recap

- OpenMP runtime call deduplication
- Infrastructure for improved OpenMP-specific feedback (remarks and more)
- Interprocedural tracking of (hidden) OpenMP runtime state
- OpenMP parallelism aware optimizations
- OpenMP target memory transfer optimizations
- OpenMP host-device optimizations