OpenMP

SC25 OpenMP Tech Talk Series

Accelerating with OpenMP
Latest and Upcoming Features

Kevin Sala – Postdoctoral Researcher
Lawrence Livermore National Laboratory (LLNL)

Outline

- Accelerating with OpenMP
- Static Shared Memory
- Dynamic Shared Memory
- Multidimensional Grid Programming
- Reducing Device Runtime Overhead
- Conclusions

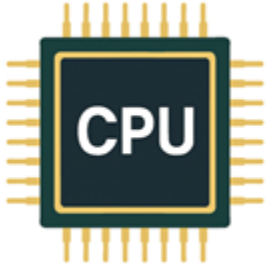
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Parallelizing with OpenMP

OpenMP

```
#pragma omp parallel for collapse(3)
for (int z = 0; z < N; z++)
  for (int y = 0; y < N; y++)
    for (int x = 0; x < N; x++)
      Matrix[x + y*N + z*N*N] *= Val;
```

OpenMP app



This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

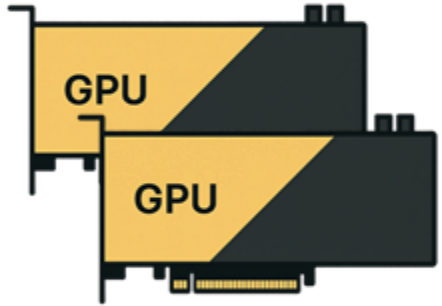
Accelerating with OpenMP?

OpenMP

```
#pragma omp parallel for collapse(3)
for (int z = 0; z < N; z++)
  for (int y = 0; y < N; y++)
    for (int x = 0; x < N; x++)
      Matrix[x + y*N + z*N*N] *= Val;
```

?

OpenMP app



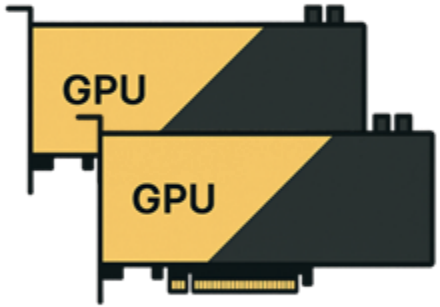
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Accelerating with OpenMP

OpenMP

```
#pragma omp target parallel for collapse(3)
for (int z = 0; z < N; z++)
  for (int y = 0; y < N; y++)
    for (int x = 0; x < N; x++)
      Matrix[x + y*N + z*N*N] *= Val;
```

OpenMP app



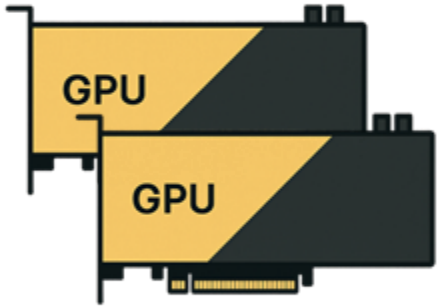
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

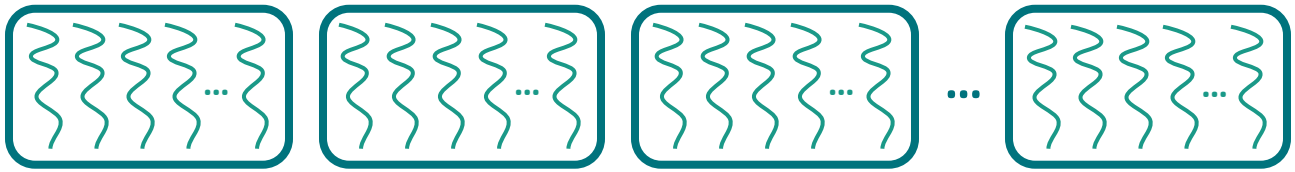
Accelerating with OpenMP

OpenMP

```
#pragma omp target teams distribute parallel for \
    map(tofrom: Matrix[0:N*N*N]) collapse(3)
for (int z = 0; z < N; z++)
  for (int y = 0; y < N; y++)
    for (int x = 0; x < N; x++)
      Matrix[x + y*N + z*N*N] *= Val;
```

OpenMP app





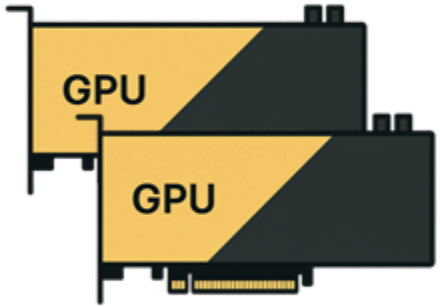
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

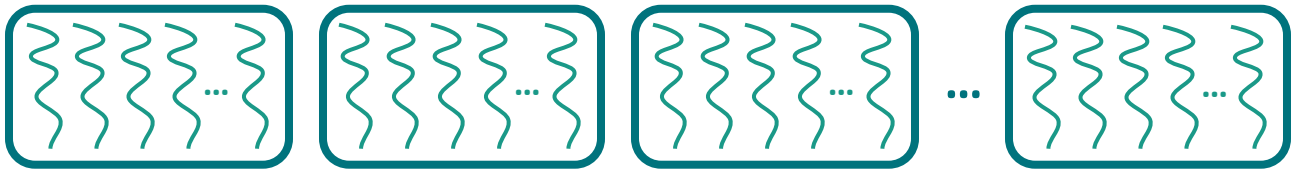
Accelerating with OpenMP

OpenMP

```
#pragma omp target teams distribute parallel for \
    num_teams(N*N*N/256) thread_limit(256)       \
    map(tofrom: Matrix[0:N*N*N]) collapse(3)
for (int z = 0; z < N; z++)
  for (int y = 0; y < N; y++)
    for (int x = 0; x < N; x++)
      Matrix[x + y*N + z*N*N] *= Val;
```

OpenMP app





This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Accelerating with Native APIs

CUDA app

HIP app

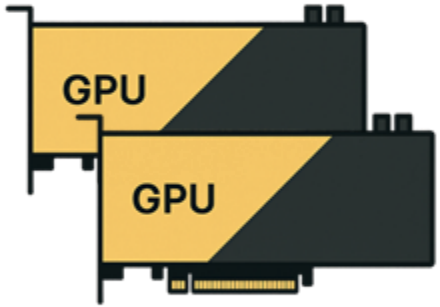
```
__global__ kernel(double *matrix, size_t N, size_t S) {
  __shared__ double tmp1[512];
  __shared__ double tmp2[];

  int x = blockIdx.x * blockDim.x + threadIdx.x;
  int y = blockIdx.y * blockDim.y + threadIdx.y;
  int z = blockIdx.z * blockDim.z + threadIdx.z;

  init(tmp1, 512);
  init(tmp2, S);
  __syncthreads();

  if (x < N && y < N && z < N)
    matrix[x + y * N + z * N * N] = ... + tmp1[...] + tmp2[...];
}

dim3 nblocks(N/256,N/256,N/256);
dim3 nthreads(256,256,256);
kernel<<<nblocks,nthreads,S*sizeof(double)>>>(matrix, N, S);
```

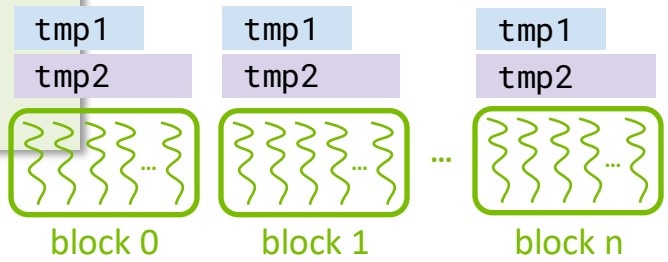
Shared memory

Single program, multiple data (SPMD)

3-D grids and blocks

Dynamic shared memory





This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

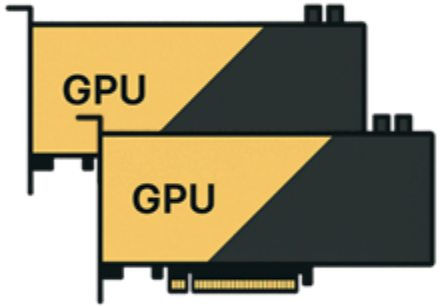
Accelerating with OpenMP

- OpenMP should expose such features to
  - Close the **performance gap** with native APIs
  - Become a **stronger alternative** to native APIs
  - Reduce application **porting effort**

OpenMP

?

OpenMP app



This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

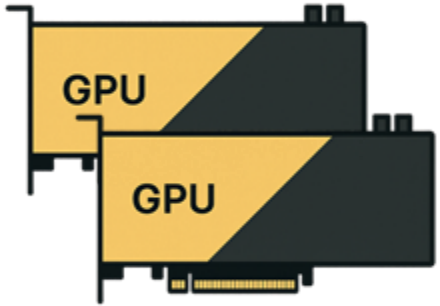
Accelerating with OpenMP

- OpenMP should expose such features to
  - Close the **performance gap** with native APIs
  - Become a **stronger alternative** to native APIs
  - Reduce application **porting effort**

OpenMP

Let's review them!

OpenMP app



This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Outline

- Accelerating with OpenMP
- **Static Shared Memory**
- Dynamic Shared Memory
- Multidimensional Grid Programming
- Reducing Device Runtime Overhead
- Conclusions

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Grouppprivate Directive

- Specify variables with static storage duration as grouppprivate
  - Each *contention group* gets its own **copy**
  - File-scope, namespace-scope or static block-scope variables

- Implementation can use **static shared memory** on the GPU

Contention group: All implicit tasks and their descendent tasks that are generated in an implicit parallel region, *R*, and in all nested regions for which *R* is the innermost enclosing implicit parallel region.

C++

```
void func(int *sum, int tid) {
  static int tmp[1000];
  #pragma omp grouppprivate(tmp)

  #pragma omp for
  for (int i = 0; i < 1000; i++)
    tmp[i] = tid + i;

  #pragma omp for reduction(+: sum)
  for (int i = 0; i < 1000; i++)
    sum += tmp[i];
}

int main() {
  int sums[10];

  #pragma omp target teams num_teams(10) thread_limit(256)
  #pragma omp parallel
  func(&sums[omp_get_team_num()], omp_get_thread_num());
}
```

Available OpenMP 6.0

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Grouppprivate Directive

- Similar to CUDA / HIP

C++

```
__global__ void func(int *sum, size_t N) {
  __shared__ int tmp[1000];
  ...
}

func<<<10,256>>>(sum, N);
```

C++

```
void func(int *sum, int tid) {
  static int tmp[1000];
  #pragma omp grouppprivate(tmp)

  #pragma omp for
  for (int i = 0; i < 1000; i++)
    tmp[i] = tid + i;

  #pragma omp for reduction(+: sum)
  for (int i = 0; i < 1000; i++)
    sum += tmp[i];
}

int main() {
  int sums[10];

  #pragma omp target teams num_teams(10) thread_limit(256)
  #pragma omp parallel
  func(&sums[omp_get_team_num()], omp_get_thread_num());
}
```

Available OpenMP 6.0

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Outline

- Accelerating with OpenMP
- Static Shared Memory
- **Dynamic Shared Memory**
- Multidimensional Grid Programming
- Reducing Device Runtime Overhead
- Conclusions

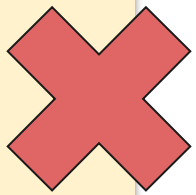
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Dynamic Grouppprivate Clause

- **Not allowed** with variable length array (VLA)
  - Static variables can't be VLA

C++

```
void func(int *sum, int tid, int N) {
  static int tmp[N];
  #pragma omp groupprivate(tmp)

  #pragma omp for
  for (int i = 0; i < N; i++)
    tmp[i] = tid + i;

  #pragma omp for reduction(+: sum)
  for (int i = 0; i < N; i++)
    sum += tmp[i];
}

int main(int argc, char **argv) {
  int N = atoi(argv[1]);
  int sums[10];

  #pragma omp target teams num_teams(10) thread_limit(256)
  #pragma omp parallel
  func(&sums[omp_get_team_num()], omp_get_thread_num(), N);
}
```



This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Dynamic Groupprivate Clause

- Request a groupprivate buffer with size determined at runtime
  - Each *contention group* gets its own **copy**
  - Available on target and teams
- Implementation can use **dynamic shared memory** on the GPU

Contention group: All implicit tasks and their descendent tasks that are generated in an implicit parallel region, *R*, and in all nested regions for which *R* is the innermost enclosing implicit parallel region.

C++

```
void func(int *sum, int tid, int N) {
  int *tmp = omp_get_dyn_groupprivate_ptr();

  #pragma omp for
  for (int i = 0; i < N; i++)
    tmp[i] = tid + i;

  #pragma omp for reduction(+: sum)
  for (int i = 0; i < N; i++)
    sum += tmp[i];
}

int main(int argc, char **argv) {
  int N = atoi(argv[1]);
  int sums[10];

  #pragma omp target teams dyn_groupprivate(N*sizeof(int)) \
              num_teams(10) thread_limit(256)
  #pragma omp parallel
  func(&sums[omp_get_team_num()], omp_get_thread_num(), N);
}
```

Approved OpenMP 6.1

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Dynamic Groupprivate Clause

- Similar to CUDA / HIP

C++

```
__global__ void func(int *sum, size_t N) {
  __shared__ int tmp[];
  ...
}

func<<<10,256,N*sizeof(double)>>>(sum, N);
```

C++

```
void func(int *sum, int tid, int N) {
  int *tmp = omp_get_dyn_groupprivate_ptr();

  #pragma omp for
  nt i = 0; i < N; i++)
  i] = tid + i;

  a omp for reduction(+: sum)
  nt i = 0; i < N; i++)
  += tmp[i];
}

int main(int argc, char **argv) {
  int N = atoi(argv[1]);
  int sums[10];

  #pragma omp target teams dyn_groupprivate(N*sizeof(int)) \
              num_teams(10) thread_limit(256)
  #pragma omp parallel
  func(&sums[omp_get_team_num()], omp_get_thread_num(), N);
}
```

Approved
OpenMP 6.1

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Dynamic Groupprivate Clause

- The fallback modifier determines what to do when **no enough** groupprivate memory to satisfy the request
  - **Abort** the execution
  - Return **null** when requesting pointer
  - Use memory from a **default memspace** (e.g., global) for each contention group

default_mem if not specified

```
#pragma omp target dyn_groupprivate(fallback(abort): N)
{
}
```

```
#pragma omp target dyn_groupprivate(fallback(null): N)
{
}
```

```
#pragma omp target dyn_groupprivate(fallback(default_mem): N)
{
}
```

```
#pragma omp target dyn_groupprivate(N)
{
}
```

Approved OpenMP 6.1

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Dynamic Groupprivate Clause

- Routine to get pointer to the groupprivate buffer

only cgroup supported for now

C++

```
#pragma omp target teams dyn_groupprivate(N)
#pragma omp parallel
{
  double *tmp = omp_get_dyn_groupprivate_ptr();
  tmp[...] = ...;
}
```

C / C++

```
void *omp_get_dyn_groupprivate_ptr(
    size_t offset = 0, int *is_fallback = NULL,
    omp_access_t access_group = omp_access_cgroup);
```

Fortran

```
type (c_ptr) function omp_get_dyn_groupprivate_ptr(&
    offset, is_fallback, access_group)
  use, intrinsic :: iso_c_binding, only : c_ptr, c_size_t
  integer (kind=c_size_t), value, optional :: offset
  logical, intent(out), optional :: is_fallback
  integer (kind=omp_access_kind), value, optional :: access_group
```

Approved OpenMP 6.1

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Dynamic Groupprivate Clause

- Routine to get size of the groupprivate buffer

C++

```
#pragma omp target teams dyn_groupprivate(N)
#pragma omp parallel
{
  size_t size = omp_get_dyn_groupprivate_size();
  if (size != 0)
    ...
}
```

C / C++

```
size_t omp_get_dyn_groupprivate_size(
    omp_access_t access_group = omp_access_cgroup);
```

Fortran

```
integer (kind=c_size_t) function omp_get_dyn_groupprivate_size(&
    access_group)
  use, intrinsic :: iso_c_binding, only : c_size_t
  integer (kind=omp_access_kind), value, optional :: access_group
```

Approved OpenMP 6.1

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Outline

- Accelerating with OpenMP
- Static Shared Memory
- Dynamic Shared Memory
- **Multidimensional Grid Programming**
- Reducing Device Runtime Overhead
- Conclusions

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Multidimensional Grid Programming

- Many accelerated applications follow multidimensional grid programming

CUDA / HIP

```
__global__ kernel(double *matrix, size_t N) {
  int x = blockIdx.x * blockDim.x + threadIdx.x;
  int y = blockIdx.y * blockDim.y + threadIdx.y;
  int z = blockIdx.z * blockDim.z + threadIdx.z;

  if (x < N && y < N && z < N)
    matrix[x + y * N + z * N * N] = ...;
}

dim3 nblocks(N/256,N/256,N/256);
dim3 nthreads(256,256,256);
kernel<<<nblocks,nthreads>>>(matrix, N);
```

Single Program,
Multiple Data (SPMD)

3-D grids and blocks

Under Discussion

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Multidimensional Grid Programming

- Many accelerated applications follow multidimensional grid programming

CUDA / HIP

```
__global__ kernel(double *matrix, size_t N) {
  int x = blockIdx.x * blockDim.x + threadIdx.x;
  int y = blockIdx.y * blockDim.y + threadIdx.y;
  int z = blockIdx.z * blockDim.z + threadIdx.z;

  if (x < N && y < N && z < N)
    matrix[x + y * N + z * N * N] = ...;
}

dim3 nblocks(N/256,N/256,N/256);
dim3 nthreads(256,256,256);
kernel<<<nblocks,nthreads>>>(matrix, N);
```

OpenMP 6.0

```
#pragma omp target teams distribute parallel for \
    num_teams(N*N*N/256) thread_limit(256) collapse(3)
for (int z = 0; z < N; ++z)
  for (int y = 0; y < N; ++y)
    for (int x = 0; x < N; ++x)
      matrix[x + y * N + z * N * N] = ...;
}
```

Under Discussion

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Multidimensional Grid Programming

- Allow **multidimensional** OpenMP teams and leagues

CUDA / HIP

```
__global__ kernel(double *matrix, size_t N) {
  int x = blockIdx.x * blockDim.x + threadIdx.x;
  int y = blockIdx.y * blockDim.y + threadIdx.y;
  int z = blockIdx.z * blockDim.z + threadIdx.z;

  if (x < N && y < N && z < N)
    matrix[x + y * N + z * N * N] = ...;
}

dim3 nblocks(N/256,N/256,N/256);
dim3 nthreads(256,256,256);
kernel<<<nblocks,nthreads>>>(matrix, N);
```

Future OpenMP

```
#pragma omp target teams num_teams(dims(3): N/256,N/256,N/256) \
    thread_limit(dims(3): 256,256,256)
#pragma omp parallel
{
  int x = omp_get_team_num_dim(1) * omp_get_num_teams_dim(1)
      + omp_get_thread_num_dim(1);
  int y = omp_get_team_num_dim(2) * omp_get_num_teams_dim(2)
      + omp_get_thread_num_dim(2);
  int z = omp_get_team_num_dim(3) * omp_get_num_teams_dim(3)
      + omp_get_thread_num_dim(3);

  matrix[x + y * N + z * N * N] = ...;
}
```

Under Discussion

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

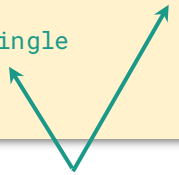
Multidimensional Grid Programming

Future OpenMP Grid Programming

```
#pragma omp target teams num_teams(dims(3): N/256,N/256,N/256) \
    thread_limit(dims(3): 256,256,256)
#pragma omp parallel
{
  int x = omp_get_team_num_dim(1) * omp_get_num_teams_dim(1)
      + omp_get_thread_num_dim(1);
  int y = omp_get_team_num_dim(2) * omp_get_num_teams_dim(2)
      + omp_get_thread_num_dim(2);
  int z = omp_get_team_num_dim(3) * omp_get_num_teams_dim(3)
      + omp_get_thread_num_dim(3);

  matrix[x + y * N + z * N * N] = ...;
}
```

Compatibility with **unidimensional** world

```
#pragma omp target teams num_teams(dims(3): N/256,N/256,N/256) \
    thread_limit(dims(3): 256,256,256)
#pragma omp parallel
{
  int team = omp_get_team_num();
  int thread = omp_get_thread_num();
  int nthreads = omp_get_num_threads();

  #pragma omp single
  { ... }
}
```



Automatic **flattening** of identifiers/sizes in constructs/routines that are not designed for the multidimensional space (or not supported yet)

Under Discussion

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Outline

- Accelerating with OpenMP
- Static Shared Memory
- Dynamic Shared Memory
- Multidimensional Grid Programming
- **Reducing Device Runtime Overhead**
- Conclusions

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Reducing Device Runtime Overhead

- The OpenMP **device runtime** can be a source of **overhead** in target regions
  - State of internal control variables (ICV)
  - Other runtime state not related to ICV

- Where does the overhead come from?
  - **Extra kernel code** due to the runtime code
  - Necessary **runtime state**, even if it may not be actually used
  - May **prevent optimizations** of the user kernel code

- Native APIs have **minimal** to none device runtime overhead

Under Discussion

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Reducing Device Runtime Overhead

- OpenMP is considering **opt-in** methods to **minimize** device overhead
  - Disallow constructs/routines that use ICV (too restrictive)
  - Allow ICV within target regions as immutable
  - Allow ICV within target regions as **immutable**, and the **compiler** can decide/compute their values

- For instance, a new clause in the target construct

Under Discussion

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Outline

- Accelerating with OpenMP
- Static Shared Memory
- Dynamic Shared Memory
- Multidimensional Grid Programming
- Reducing Device Runtime Overhead
- **Conclusions**

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

Conclusions

- **Available** and **upcoming** features for accelerating code
  - groupprivate directive in OpenMP 6.0
  - dyn_groupprivate clause approved for OpenMP 6.1
  - Multidimensional teams and leagues under discussion
  - Reducing target runtime overhead under discussion

- Why?
  - **Closing** the **performance gap** with native models
  - **Porting** accelerated applications becomes **simpler**
  - Using **low-level features** but still on a widely-supported standard

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-2013109.

SC25 OpenMP Tech Talk Series

openmp.org — OpenMP API specs, forum, reference guides, and more

link.openmp.org/sc25talks — SC25 OpenMP Tech Talk videos and presentations