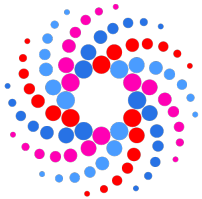
OpenMP®

SC'22 Booth Talk Series



SC22

Dallas, TX | hpc accelerates.

The Lovely loop Construct:
Declarative Semantics in OpenMP

Tim Mattson, Senior Principal Engineer, Intel Corp.

Check out our OpenMP music videos:

- https://www.youtube.com/watch?v=5ePyCol1cpw
- https://www.youtube.com/watch?v=9bHgtzIeQL0

Legal Disclaimer & Optimization Notice

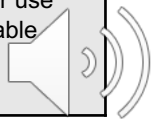
This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.
Notice revision #20110804



Third party names are the property of their owners

Disclaimer

- The views expressed in this talk are those of the speaker and not his employer.

- If I say something "smart" or worthwhile:
  - Credit goes to the many smart people I work with.

- If I say something stupid…
  - It's my own fault

I work in Intel's research labs. I don't build products. Instead, I get to poke into dark corners and think silly thoughts… just to make sure we don't miss any great ideas.

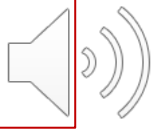
Hence, my views are by design far "off the roadmap".

Explicit semantics for directive driven programming

- OpenMP is **Wonderful**
  - With OpenMP's directives, I tell the system just want I want it to do.
  - The directives are **Explicit** … I control the number of threads, how loops are distributed between threads, affinity, and much more.
  - Control is a good thing when you want to get the most from a particular system.

```
#pragma omp parallel proc_bind(spread) num_threads(16)
{
   #pragma omp single nowait
         printf("number of threads = %d\n",omp_get_num_threads());

   #pragma omp for schedule(monotonic:dynamic,5) collapse(2) private(j) reduction(+:huh)
   for(i=0;i<N;i++)
      for (j=0;j<M;j++){
         huh+=func(i,j,N);
      }
}
```



Explicit semantics for directive driven programming

- OpenMP is Wonderful
  - With OpenMP's directives, I tell the system just want I want it to do.
  - The directives are Explicit ... I control the number of threads, how loops are distributed between threads, affinity, and much more.
  - Control is a good thing when you want to get the most from a particular system.

- OpenMP is a **pain in the Ass**
  - OpenMP's directives are too complicated … I have to know too much about the hardware
  - Why can't I just tell the system "what" I want done, not "how" to do it.
  - Control is a bad thing when you want portability across systems

You think it's bad for multithreading … look what happens when you go to a GP[illegible]!!

Solution: parallel stencil (heat)

Explicit Semantics

```
// Compute the next timestep, given the current timestep
void solve(const int n, const double alpha, const double dx, const double dt, const double
* restrict u, double * restrict u_tmp) {
  const double r = alpha * dt / (dx * dx);
  const double r2 = 1.0 - 4.0*r;

  // Loop over the nxn grid
  #pragma omp target map(tofrom: u[0:n*n], u_tmp[0:n*n])
  #pragma omp teams distribute parallel for simd
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {

      // Update the 5-point stencil, using boundary conditions on the edges of the domain.
      // Boundaries are zero because the MMS solution is zero there.
      u_tmp[i+j*n] =  r2 * u[i+j*n] +
      r * ((i < n-1) ? u[i+1+j*n] : 0.0) +
      r * ((i > 0)   ? u[i-1+j*n] : 0.0) +
      r * ((j < n-1) ? u[i+(j+1)*n] : 0.0) +
      r * ((j > 0)   ? u[i+(j-1)*n] : 0.0);
}}}
```

The Big Ugly Directive (this BUD's for you)

The Loop construct: OpenMP's first declarative construct

- The loop construct (added with OpenMP 5.0) tells the system to parallelize the loop as it chooses. It is **declarative** …. It says nothing about how the compiler should parallelize the loop

```
#pragma omp loop
For(i=0;i<N;i++)
{
    //loop body
}
```

- The loop construct tells the OpenMP compiler/runtime:
  - The loop iterations can run in any order … the program is correct if they run concurrently.
  - The region associated with the loop construct does NOT contain any OpenMP directives or runtime library routines.

Solution: parallel stencil (heat)

Declarative Semantics

```
// Compute the next timestep, given the current timestep
void solve(const int n, const double alpha, const double dx, const double dt, const double
* restrict u, double * restrict u_tmp) {
  const double r = alpha * dt / (dx * dx);
  const double r2 = 1.0 - 4.0*r;

  // Loop over the nxn grid
  #pragma omp target map(tofrom: u[0:n*n], u_tmp[0:n*n])
  #pragma omp loop
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {

      // Update the 5-point stencil, using boundary conditions on the edges of the domain.
      // Boundaries are zero because the MMS solution is zero there.
      u_tmp[i+j*n] =  r2 * u[i+j*n] +
      r * ((i < n-1) ? u[i+1+j*n] : 0.0) +
      r * ((i > 0)   ? u[i-1+j*n] : 0.0) +
      r * ((j < n-1) ? u[i+(j+1)*n] : 0.0) +
      r * ((j > 0)   ? u[i+(j-1)*n] : 0.0);
}}}
```

The loop construct

The Loop construct: OpenMP's first declarative construct

- The behavior of the loop construct difficult to understand based on the text in the OpenMP 5.2 specification. In the specification, it says:
  - that the concurrent iterations are executed by the encountering thread.

- For use with target … that means the initial thread on the device, i.e., one thread and therefore, no parallelism.
- Here's the catch … the OpenMP spec is written for compiler writers, not applications programmers. *Any* compiler writer knows the famous "as if rule".

A compiler can apply any transformations to code during compilation as long as the transformed code makes no changes to the "observed behavior" of the program.

- Hence, the initial thread on the device can fork a league of teams, map them onto the compute units of the GPU, and run them on the compute units of a GPU.
  - The problem is the spec NEVER actually says this anywhere.

Summary

- The loop construct is declarative … it does not stipulate how a loop is to be parallelized.

- It is more portable … the system decides how to map loops onto the construct and therefore it can handle the full range of systems openMP supports.

- It is essential for GPU programming … This BUD may be for you

```
#pragma omp teams distribute parallel for simd
```

… but I prefer

```
#pragma omp loop
```

My Greenlandic skin-on-frame kayak in the middle of Budd Inlet during a negative tide

And don't forget to check out our OpenMP music videos:

- https://www.youtube.com/watch?v=5ePyCol1cpw
- https://www.youtube.com/watch?v=9bHgtzIeQL0

BUD: Big Ugly Directive