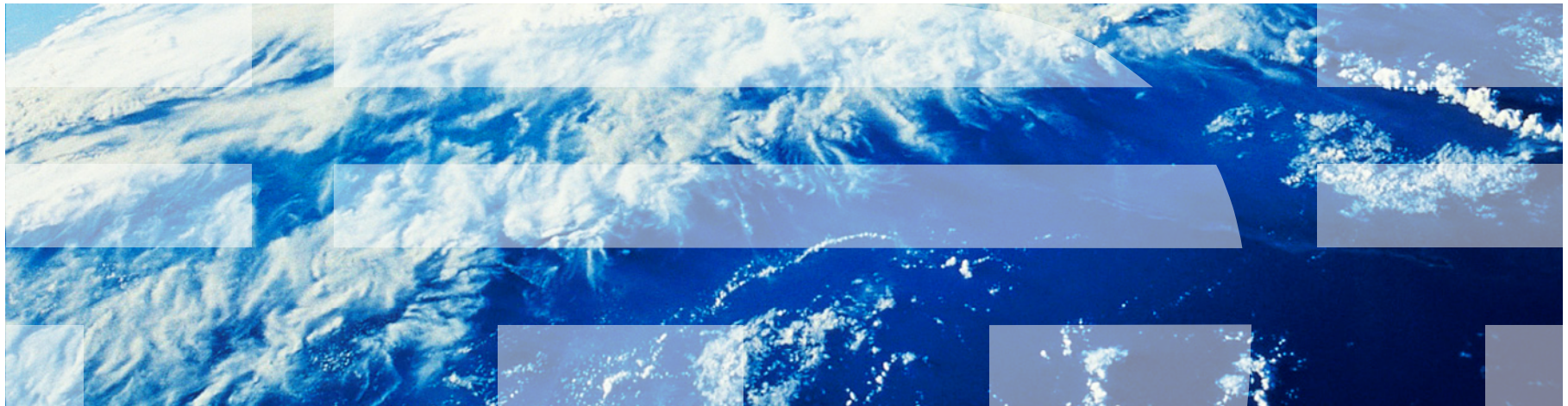


Alexandre Eichenberger

4/22/16



The Case for Enhancing Portability in Future OpenMP



T.J. Watson, IBM Research

© 2011 IBM Corporation

Overview

- **OpenMP 4.5 is a powerful tool for accelerators**
 - exposes new patterns
- **Some constructs could be better defined to enhance portability**
 - example: target teams executing on host
- **Some constructs are used in new ways, and could be relaxed**
 - example: parallel & collapsed loops
- **Take away**
 - a few small steps can greatly improve the performance portability of OpenMP

Example 1: I wrote some good target code

- Efficient code for my accelerators

parallelism

```
int devNum = MAX(1, omp_get_num_devices()); int n = N / devNum;
#pragma omp parallel for num_threads(devNum)
for (int d=0; d<devNum; d++) {
    #pragma omp target teams num_teams(1024) thread_limit(1024) device(d)
    {
        #pragma omp distribute
        for (int i=d*n; i<d*n+n; i++) {
            #pragma omp parallel for
            for (int j=0; j<M; j++) {
                // loop code for device d, loop i & j
            }
        }
    }
}
```

for each device

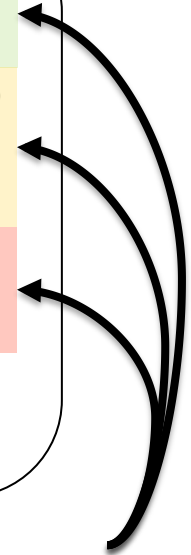
for each team

for each thread

- Now someone wants to run it on a machine without accelerators
- Or some data sets are too small to be profitable on accelerators

What could go wrong?

```
int devNum = MAX(1, omp_get_num_devices()); int n = N / devNum;
#pragma omp parallel for num_threads(devNum)
for (int d=0; d<devNum; d++) {
    #pragma omp target teams num_teams(1024) thread_limit(1024) device(d)
    {
        #pragma omp distribute
        for (int i=d*n i<d*n+n; i++) {
            #pragma omp parallel for
            for (int j=0; j<M; j++) {
                // loop code for device d, loop i & j
            }
        }
    }
}
```

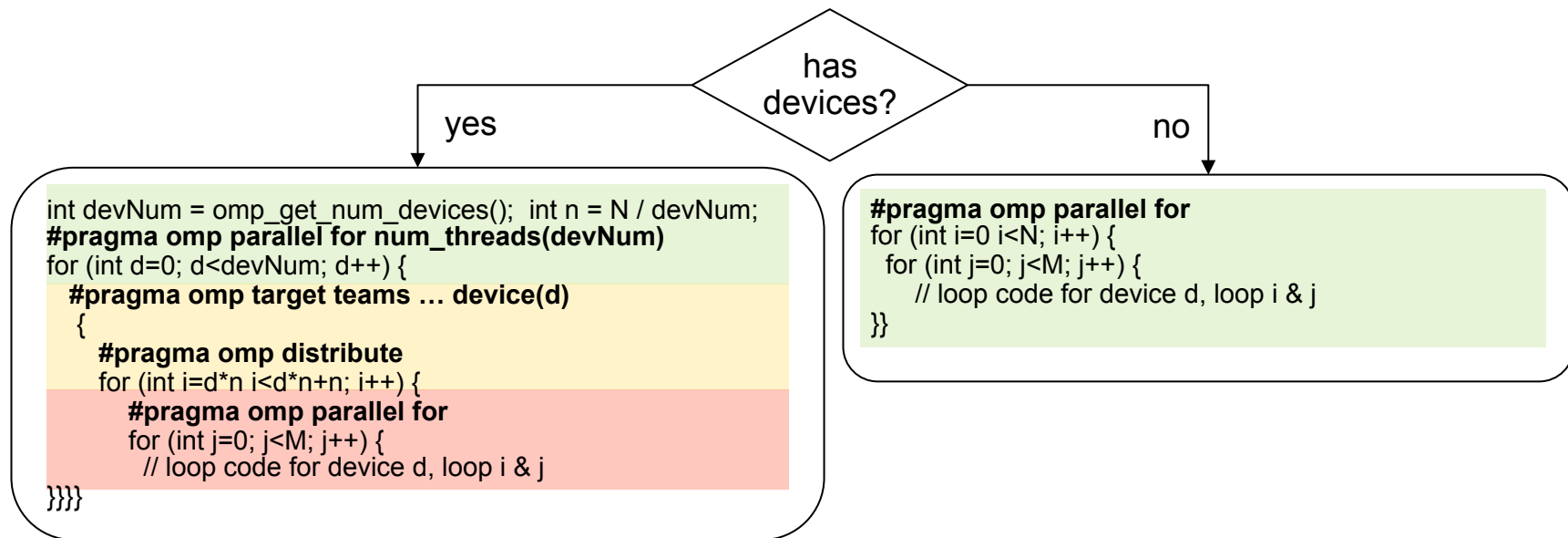


▪ Where to get the parallelism on the host?

- parallel for over devices? target teams? innermost parallel for?
- target teams behavior on the host?
 - standard does not prescribe if run in parallel or not
 - target teams is not disabled by OMP_NESTED=FALSE
- even when disabled, distribute / parallel / for are costly
 - extra runtime calls, inflexible code structures, outlining...

A user could write two versions?

- One for target devices, one for the host



- But users really don't like it
 - replicating code is a maintenance issue
 - and is against OpenMP pragma-only paradigm

A first step to help portability

▪ Iterator over all devices

- more portable to have a construct that distribute work over devices
- with predetermined behavior when no devices are available

▪ Well defined Target construct on host

- target teams become a parallel on the host
 - because coarse grain parallelism is often best
- integrated into host contention group
- integrated with the control for nested parallelism
 - controlled by *nest* & *max-active-level* ICVs
- integrated with proc-bind affinity
- ignore parameters meant for devices
 - thread limit is best for GPUs, has no role on host

▪ Allows for eliminating some constructs

- nested parallelism inspired by GPUs (teams/distribute/parallel/for)
- is not beneficial on “thread-poor” host
- compiler could recognize the “if(omp_is_initial_device())” pattern
- or could introduce custom if values: “if(onhost)” & “if(ondevice)”

More advanced extensions: “if-and-only-if”

- **May allow more than one directive per construct**
 - for the same piece of code (e.g. code to be executed on a target)
 - add one set of directive for target devices
 - add one set of directive for host device

```
...
int devNum = omp_get_num_devices();
#pragma omp target teams distribute num_teams(1024) device(d) iff(devNum)
#pragma omp parallel for iff(!devNum)
for (int i=d*n; i<d*(n+1); i++) {
    #pragma omp parallel for iff(devNum)
    for (int j=0; j<M; j++) {
        // loop code for device d, loop i & j
    }
}
```

Two mutually exclusive pragma with “if and only if”

Example 2: Increased reliance on collapsed loops

- **Typical hosts have small numbers of threads**
 - thus OpenMP 3.1 code did not use many collapsed loops
 - benefits were small (outer-loop parallelism was sufficient)
 - overhead were significant (collapse is expensive to implement)

- **Target devices have often a magnitude more threads**
 - we see many more collapsed loop in target codes
 - need much more parallelism than outer-most loop
 - bring in more by collapsing many nested loops

- **This cause a problem for portability**
 - good code for devices has more overhead for host code

A second step towards portability

- **As collapse constructs is more frequent...**
 - generate more optimized code for collapsed loop
- **May allow “onhost” or “ontarget” clause qualifier**
 - e.g. “collapse(onhost: 1, ontarget: 3)
- **Or redefine a collapse that is less descriptive**
 - as of OpenMP 4.5, it precisely describe how iterations must be collapsed

Summary

- **Implementations of OpenMP 4.5 show promising performance**
 - many codes execute nearly as fast as natively-programmed codes
- **When defining the standard, not all performance porting patterns were clear**
- **With what we know, we should be able to address many of these issues at the OpenMP level by relatively minor tweaks**