NUMA aware DPC++/SYCL extensions for CPUs and X^e GPUs

oneAPI DPC++/SYCL Technical Advisory Board Meeting

December 16th, 2020

Notices

DISTRIBUTION STATEMENT: None Required

Disclosure Notice: This presentation is bound by Non-Disclosure Agreements between Intel Corporation, the Department of Energy, and DOE National Labs, and is therefore for Internal Use Only and not for distribution outside these organizations or publication outside this Subcontract. This document may be distributed only to those persons listed on the most recent Multi Party Authorization for the A21 program with a business need for access to the document.

Intel Proprietary Information: This document contains trade secrets and/or proprietary information of Intel Corporation and Intel Federal LLC ("Intel") and is exempt from disclosure under the Freedom of Information Act. The information contained herein shall not be duplicated, used or disclosed outside the U.S. Department of Energy, Lawrence Livermore National Security LLC (LLNS), UChicago Argonne LLC, Los Alamos National Security LLC (LANS), National Technology and Engineering Solutions of Sandia LLC (NTESS), or UT-Battelle LLC, except as permitted by Intel's CORAL A21 Subcontract No. 8F-30005. The data subject to this restriction are contained in all sheets of this document.

USG Disclaimer: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Export: This document contains information that is subject to export control under the Export Administration Regulations. However the contents remain within the applicable ECCN's provided in the most recent Multi Party Authorization that is applicable to the CORAL A21 Program.

Intel Disclaimer: Intel makes available this document and the information contained herein in furtherance of CORAL. None of the information contained therein is, or should be construed, as advice. While Intel makes every effort to present accurate and reliable information, Intel does not guarantee the accuracy, completeness, efficacy, or timeliness of such information. Use of such information is voluntary, and reliance on it should only be undertaken after an independent review by qualified experts.

Access to this document is with the understanding that Intel is not engaged in rendering advice or other professional services. Information in this document may be changed or updated without notice by Intel.

This document contains copyright information, the terms of which must be observed and followed.

Reference herein to any specific commercial product, process or service does not constitute or imply endorsement, recommendation, or favoring by Intel or the US Government.

Intel makes no representations whatsoever about this document or the information contained herein. IN NO EVENT SHALL INTEL BE LIABLE TO ANY PARTY FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES FOR ANY USE OF THIS DOCUMENT, INCLUDING, WITHOUT LIMITATION, ANY LOST PROFITS, BUSINESS INTERRUPTION, OR OTHERWISE, EVEN IF INTEL IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

DPC++/SYCL: NUMA, Affinity and Scheduling

- NUMA domain and affinity control
  - Queue: traits / properties (CPU: places = numa_domains, sockets, cores, threads, etc., GPU: places = device, tile, slice, EU, EU threads, etc.)
  - Kernel: C++ attribute for affinity (e.g. master, spread, close)
- Global and kernel level control of scheduling types
  - static, dynamic, guided, runtime scheduling
    - chunk size (represented as local group size?)
  - user-defined scheduling type
- Open question: what would be a proper level control of thread-data (or task-data) affinity (coarse, fine or both?) for DPC++/SYCL programmers?

NUMA Topic: RM1 and RM2 Discussion Notes

• We may need high abstraction such as "spread" and "close" for programmers. • We may also need to support fine-level control for ninja programmers with a good mirror to architectural hierarchy.	• Kokkos primarily uses OpenMP environment variables to get ~10x performance for some Kokkos users. • Places (an abstraction) is a reasonable abstraction for NUMA affinity control • Good thread-affinity control is tied to implementation specifics	• How to present NUMA control / usage model to users is very important for ease of use.	• A big customer prefers a simpler method for applications w.r.t. NUMA domains usage. • User expects implicit NUMA-aware support for applications cross-tile.
• TensorFlow uses and supports a high-level control of NUMA domains for TF performance.	• How to support NUMA control has impact on portability and scheduling. Explicit NUMA control serves better from applications in general. • make subdevice (tile) as a GPU (a NUMA domain), then, the scheduling happens in the tile, which does minimizel NUMA impact, a bit more work for users. • I understand why people want an easy mode, and if we can give people an easy mode that *works* then I'm all for it. I think Xinmin's point about tying data to tasks is key: if we can design something where programmers say "Here are my data dependencies, please schedule this in a way that gets good performance" we'll have more luck than if we ask non-experts to reason about things like whether pages should be interleaved and the granularity of thread scheduling.	• DPC++ (Gold) started with a high level control DPCPP_CPU_CU_AFFINITY={master | close | spread} for CPU. There are scheduling implications as well for thread-data affinity or task-data affinity. • GPU (HW and driver) may support a "fixed mode" for programmers on NUMA thread-data affinity control. • C++ standard committee executor WG is investigating NUMA support as well.	

NUMA Places for Thread-Data Affinity

- DPCPP_CPU_PLACES = {sockets | numa_domains | cores | threads} specifies the places where threads are pinned, which is analogous to OMP_PLACES in OpenMP. Threads within a place can migrate among the CPU cores of that place. Default value is "cores".

sockets	Each place is a single socket which consists of one or multiple cores.
numa_domains	Each place is a single NUMA node.
cores	Each place is a single CPU core which has one or multiple hardware threads.
threads	Each place is a single hardware thread.

- When DPCPP_CPU_PLACES is set to numa_domains and threads are bound to NUMA nodes.
- SYCL nd_range is uniformly distributed to numa nodes.
- Thread-data affinity are required in order to achieve high performance, so it is important that algorithm is NUMA aware and satisfy the NUMA requirement.
- For data locality, it needs memory first-touch, in order to prevent cross-NUMA node memory access which has much higher latency than within-NUMA node memory access. Applications with thread-data affinity or locality can benefit significantly from NUMA API.

Thread Binding: Affinity

- DPCPP_CPU_CU_AFFINITY = {close | spread | master} sets how threads are pinned to CPU cores based on thread index.

close	Threads are pinned to CPU cores successively through available spaces.
spread	Threads are spread (evenly scattered) to available places.
master	Threads are put in the same places as master.

- This is analogous to OMP_PROC_BIND in OpenMP.
- By default, this env variable is not set and all threads, except for master thread, are pinned to CPU cores according to their index in TBB.
- If this env is set, master thread is pinned as well.
- Spread could be helpful if adjacent threads don't share resources or DPCPP_CPU_NUM_CUS is set to the number of physical CPU cores.

Use SYCL and OpenMP Subdevices

- SYCL Explicit Subdevice Usage
  - auto SubDevices = dev.create_sub_devices<sycl::info::partition_property::partition_by_affinity_domain>(sycl::info::partition_affinity_domain::numa);
- OpenMP Subdevice clause extension
  - subdevice(subdevice_level, sdev_id : num_subdevices: sdev_stride)
  - #pragma omp target parallel for device(id) subdevice(sd_level, sd_id)
  - Leverage OpenMP “Places”
    - subdevice level 0: Tile subdevice level 1: Slice

C/C++ OpenMP Intel Subdevice Usage Example

```
/// Test offloading to multiple devices
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define N 16384

int main() {
  int numSubDevices = 2;
  int numThreads = 4 * numSubDevices;
  int *a = new int[N]; int *b = new int[N]; int *c = new int[N];
  for (int i = 0; i < N; i++) {
    a[i] = i; b[i] = N - i;
  }
  printf("Testing offload with %d subdevices, %d threads\n", numSubDevices, numThreads);
  int deviceId = omp_get_default_device();

#pragma omp parallel num_threads(numThreads)
  {
    int threadNum = omp_get_thread_num();
    int subId = threadNum % numSubDevices;
    int begin = threadNum * N / numThreads;
    int count = N / numThreads;
#pragma omp target parallel for device(deviceId) subdevice(0, subId) \
                   map(a[begin:count]) map(b[begin:count]) map(c[begin:count])
    for (int i = 0; i < count; i++)
      c[begin + i] = a[begin + i] + b[begin + i];
  }

  for (int i = 0; i < N; i++) {
    if (N != c[i]) {
      printf("FAIL\n"); return EXIT_FAILURE;
    }
  }
  printf("PASS\n");
  delete[] a; delete[] b; delete[] c;
  return EXIT_SUCCESS;
}
```