Planning for the CORAL LLNL Sierra Supercomputer and Locality-Aware Memory Association for Multi-Target Worksharing in OpenMP

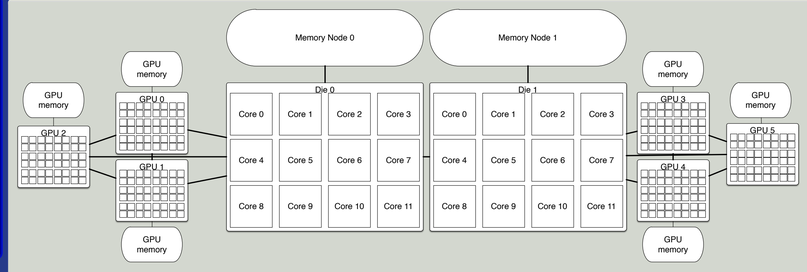
Thomas R. W. Scogland* Bronis R. de Supinski*
scogland1@llnl.gov bronis@llnl.gov
*Lawrence Livermore National Laboratory, Livermore, CA 94551 †Department of Computer Science, Virginia Tech, Blacksburg, VA 24060 USA. USA

SciDAC-3 CalLat

Sierra - CORAL at LLNL

- The hardware:
  - CPU: Multiple IBM Power 9 CPUs
  - Accelerators: Multiple NVIDIA GPUs per CPU
  - Interconnect: Coherent NVLINK/CAPI interface between CPUs and local GPUs
- Programming model: MPI+X, where X is OpenMP, CUDA or QUDA
- New tools and models are needed to address complex multi-level systems like Sierra will be
- This is our work toward that goal

Coral Sierra Mock-Up



Main Question

How can we address hierarchical memory and multiple accelerators with a single, unified extension

Our Solution: Memory Association and Work Partitioning

- Partition a range across threads or devices
  - Parallel regions can be partitioned across threads, much like a workshared loop
  - Target for loops can be partitioned, rather than scheduled, to split a loop across target devices
- Specify the association between input, output, and a partitioned range by extending the `map` clause
  - Add a mapping type option, to support indirect and user-defined mappings
  - Bind the partitioning to a mapped variable to partition that variable along with the data
- Nest partitioned `parallel` or `target` regions to address hierarchical memory systems
- Adaptively partition to achieve load-balance across the devices

Manual Partitioning

```
1  float arr[WORK_SIZE] = {0};
2  #pragma omp parallel shared(arr)
3  {
4    int tid = omp_get_thread_num();
5    int nt = omp_get_num_threads();
6    int iters = WORK_SIZE / nt;
7    int start = tid * iters;
8    int end = start + iters;
9    do_work(start,end,arr);
10 }
```

Extended Partitioning

```
1  float arr[WORK_SIZE] = {0};
2  int start = 0;
3  int end = WORK_SIZE;
4  #pragma omp parallel map(tofrom: arr[:,id]) \
5    partition(adaptive: id=start; id<end; id++)
6  {
7    int tid = omp_get_thread_num();
8    int nt = omp_get_num_threads();
9    do_work(start,end,arr);
10 }
```

Example Usage: GEMM

```
1  float A[i_size][j_size], B[i_size][j_size];
2  float *C = (float*)malloc(sizeof(C[0])*i_size*j_size);
3  int C_stride = j_size, j_start = 0, j_end = j_size;
4  #pragma omp parallel proc_bind(spread)                \
5    num_threads(omp_get_num_places())                   \
6    partition(adaptive: j_id=j_start; j_id<j_end; ++j_id)\
7    map(to: A[0:i_size][:,j_id], B[0:i_size][0:j_size])  \
8    map(tofrom: C[0:i_size][:,j_id])
9  {
10 #pragma omp target teams distribute parallel for     \
11   devices(OMP_TYPE_ALL,*) map(to: A[:][:], B[:,i][:])\
12   partition(adaptive) map(tofrom: C[:,i,C_stride][:])
13   for (int i = 0; i < i_size; ++i) {
14 #pragma omp bind_partition(j_id) // Optional
15     for (int j = j_start; j < j_end; ++j) {
16       float sum = 0.0;
17       for (int k = 0; k < k_size; ++k) {
18         sum += A[k][j] * B[i][k];
19       }
20       C[i * C_stride + j] = sum;
21     }
22   }
23 }
```

Lines 4-5 Create one thread on each OpenMP "place" and partition the devices across them

Line 6 Partition the range j_start to j_end across devices, binding the device's range to j_id, partitioning the inner loop

Line 7 Map the B matrix in completely, partition the columns of the A matrix according to j_id

Line 8 Map the C matrix in partitioning the columns with range j_id

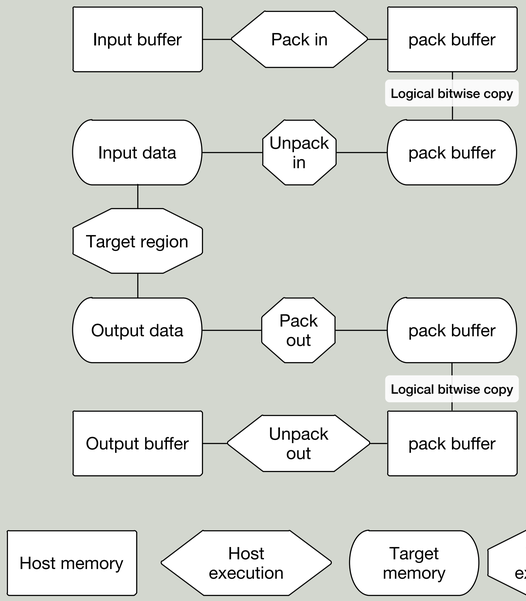
Line 11 Split this target across all devices, map in all of A from the outer partitioning and partition B by rows

Line 12 Partition the outer loop with the adaptive schedule, binding the range to i, map C in and out partitioned to match the i range with the new stride stored in C_stride

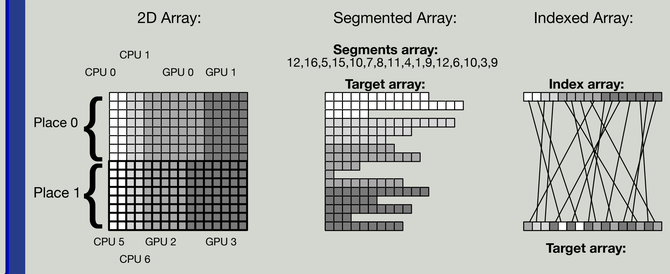
Line 14 Bind the timing of the j_id partitioning to the inner loop

Custom associations

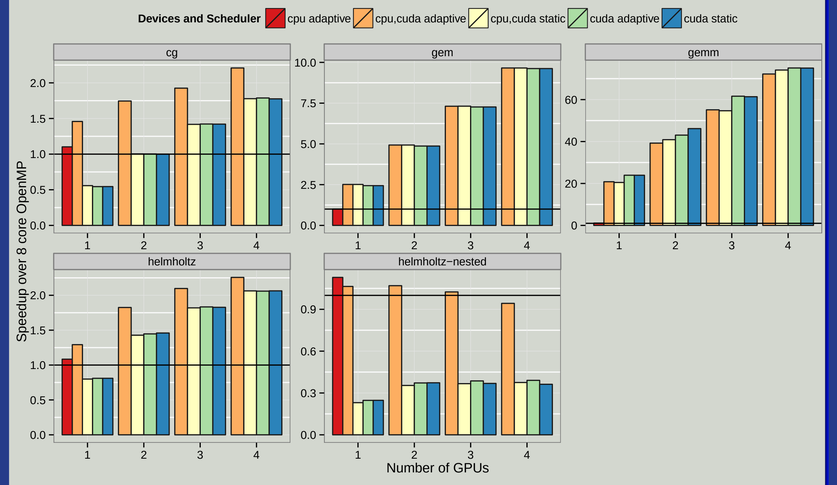
General user-defined data association support with pipelining:



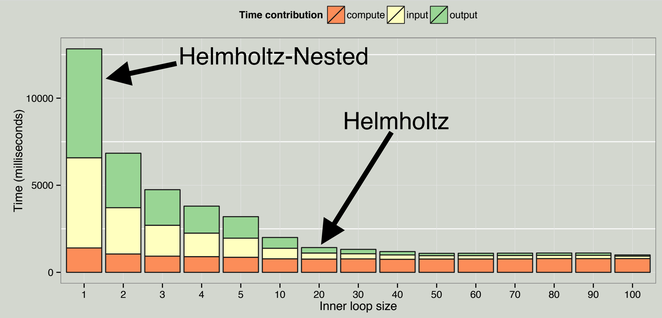
Memory Association Types



Results: Co-Scheduling Performance

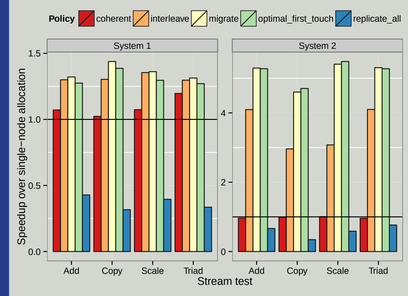


Results: Data-movement Cost of Frequent Re-Balancing

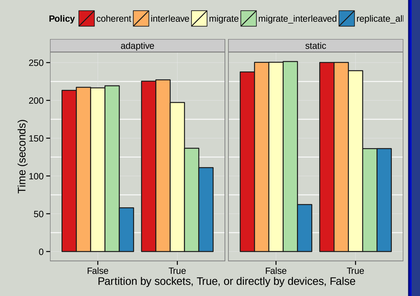


Results: Memory-Movement Optimization

Stream Bandwidth with NUMA Policies



GEMM QCD Kernel CPU-only with NUMA Policies



Pursuing Lattice QCD on Sierra and Beyond

- We are coordinating to apply this approach to Lattice QCD applications as part of the CalLat project, preparing to address the next generation of hardware not just for Sierra, but portably across the full range of next-generation supercomputers
- Memory association decouples data mapping from devices, allowing the runtime to mutate the data however is most appropriate
- Our prototype achieves up to a 50× speedup over eight core CPU with four GPUs, and we show a nearly 2× speedup for a previously averse benchmark as well

Related Papers

[1] T. R. W. Scogland, B. R. de Supinski, and W. Feng. Locality-Aware Memory Association for Multi-Target Worksharing in OpenMP. In *International Conference on Parallel Architectures and Compilation Techniques*, 2016, **Under consideration**.

[2] T. R. W. Scogland, W. Feng, B. Rountree, and B. R. de Supinski. CoreTSAR: Core task-size adapting runtime. *IEEE Transactions on Parallel and Distributed Systems*, 2014.

[3] T. R. W. Scogland, B. Rountree, W. Feng, and B. R. de Supinski. Heterogeneous Task Scheduling for Accelerated OpenMP. In *International Parallel and Distributed Processing Symposium*, pages 144–155. IEEE Computer Society, May 2012.

[4] T. R. W. Scogland, B. Rountree, W. Feng, and B. R. de Supinski. CoreTSAR: Adaptive Worksharing for Heterogeneous Systems. In *International Supercomputing Conference*, Leipzig, June 2014.

Prepared by LLNL under Contract DE-AC52-07NA27344 (LLNL-POST-674449)