CATEGORY: DEVELOPER - PROGRAMMING LANGUAGES - DL01

POSTER P5128 | CONTACT NAME Thomas Scogland: tom@scogland.com

GPU TECHNOLOGY CONFERENCE

Lawrence Livermore National Laboratory

LLNL-POST-661955

Locality-Aware Memory Association for Multi-Target Worksharing in OpenMP

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

VirginiaTech Invent the Future

Introduction

- Heterogeneity is everywhere
  - Accelerators are gaining popularity: GPUs, FPGAs, DSPs etc.
  - NUMA memory is proliferating
  - Even homogeneous systems are heterogeneous due to OS noise!
- Programming models like OpenMP 4.0 and OpenACC are being created to address heterogeneity, but *do not handle multiple devices*
- CPU models like OpenMP handle multiple devices, but *do not address hierarchical memory*

Copy Bandwidth Between Components in a Multi-GPU System

From/To	MN 0	MN 1	GPU 0	GPU 1	GPU 2	GPU 3
Memory Node (MN) 0	12,407	8,704	3,851	3,855	3,785	3,758
Memory Node (MN) 1	8,963	17,920	3,795	3,771	4,032	4,096
Interleaved	15,639	14,298	3,454	3,238	3,429	3,457
GPU 0	3,460	2,926	97,469	4,890	N/A	N/A
GPU 1	3,460	2,922	4,890	97,619	N/A	N/A
GPU 2	2,833	3,971	N/A	N/A	97,630	4,890
GPU 3	2,820	4,108	N/A	N/A	4,890	97,636

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

```
float arr[WORK_SIZE] = {0};
#pragma omp parallel shared(arr)
{
  int tid = omp_get_thread_num();
  int nt = omp_get_num_threads();
  int iters = WORK_SIZE / nt;
  int start = tid * iters;
  int end = start + iters;
  do_work(start,end,arr);
}
```

Extended Partitioning

```
float arr[WORK_SIZE] = {0};
int start = 0;
int end = WORK_SIZE;
#pragma omp parallel map(tofrom: arr[:,id]) \
  partition(adaptive: id=start; id<end; id++)
{
  int tid = omp_get_thread_num();
  int nt = omp_get_num_threads();
  do_work(start,end,arr);
}
```

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

Adaptation: Load-balancing for Partitioned Worksharing

Variables

I = total iterations available

f_j = fraction of iterations for compute unit j

p_j = recent time/iteration for compute unit j

n = number of compute devices

$t_j^+ (\text{or } t_j^-)$ = time over (or under) equal

Linear Program

$$min(\sum_{j=1}^{n-1} t_j^+ + t_j^-)$$

$$\sum_{j=0}^{n} i_j = 1$$

$$f_2 * p_2 - f_1 * p_1 = t_1^+ - t_1^-$$

$$f_3 * p_3 - f_1 * p_1 = t_2^+ - t_2^-$$

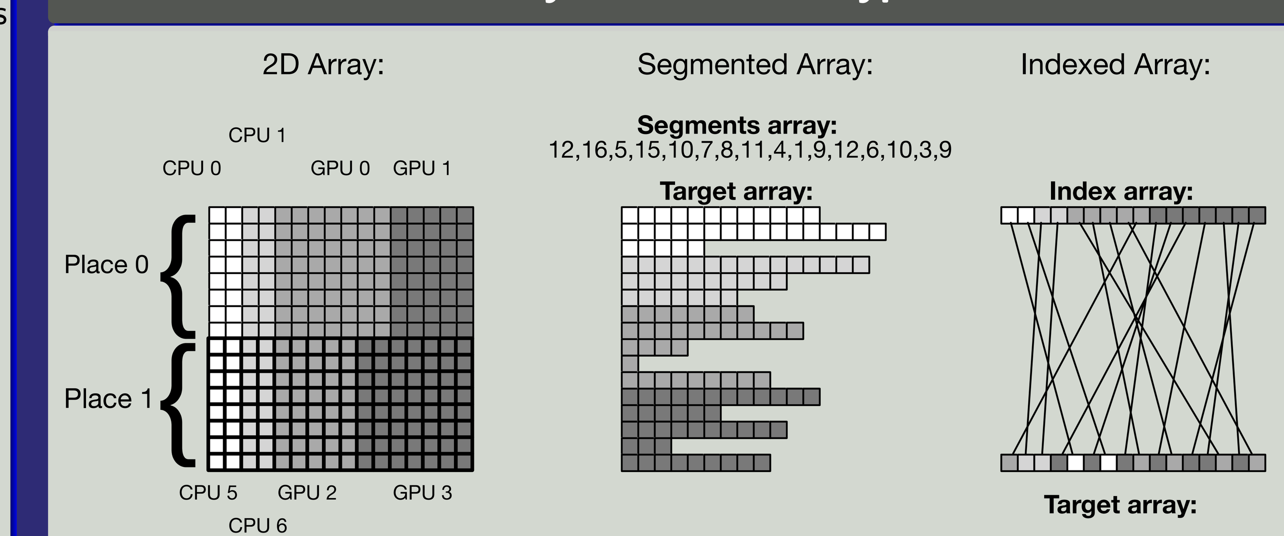
$$\vdots$$

$$f_n * p_n - f_1 * p_1 = t_{n-1}^+ - t_{n-1}^-$$

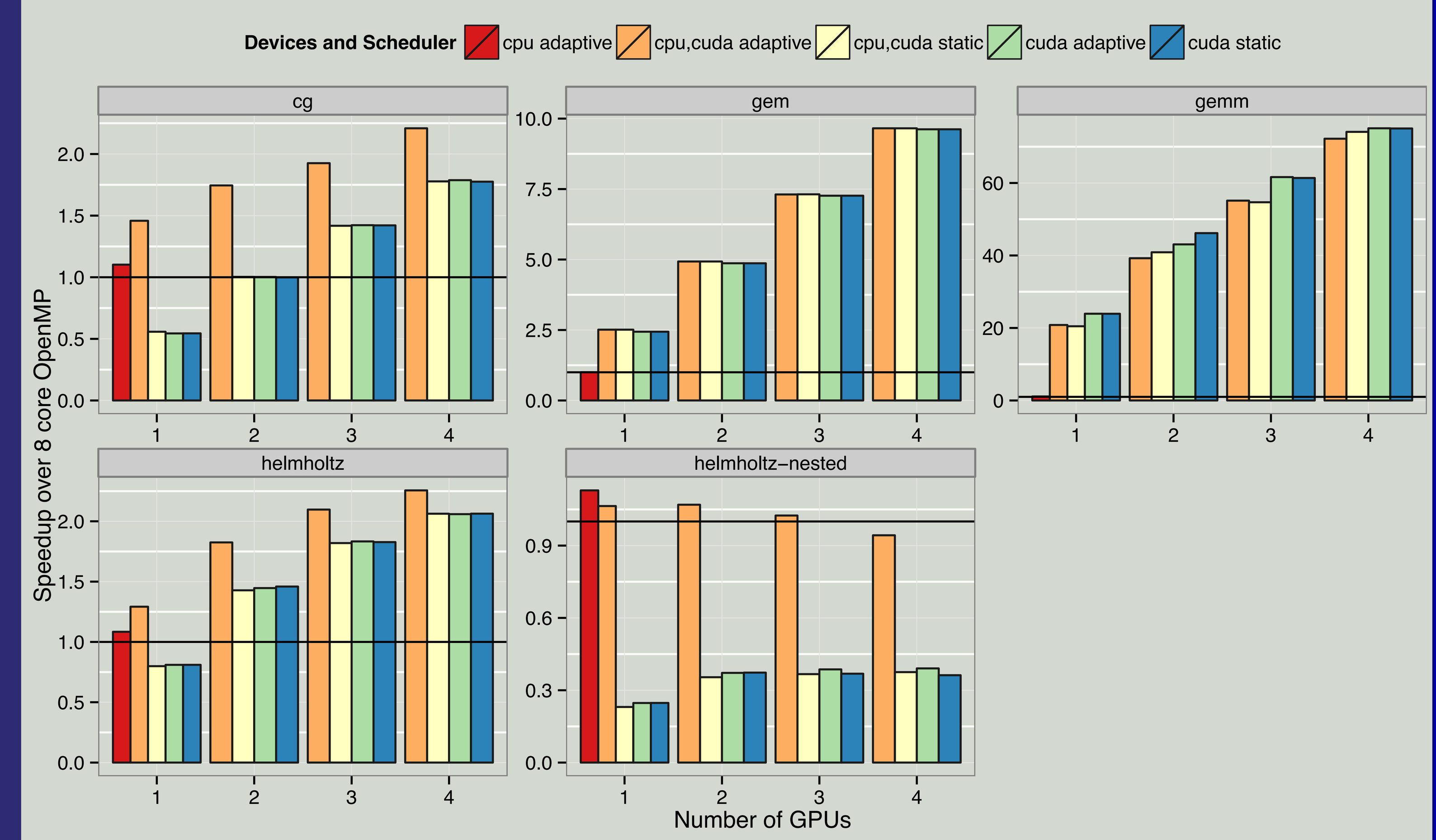
In Words

Minimize the sum of differences between each device's predicted runtime and the predicted runtime of other devices, or minimize waiting/blocking time.

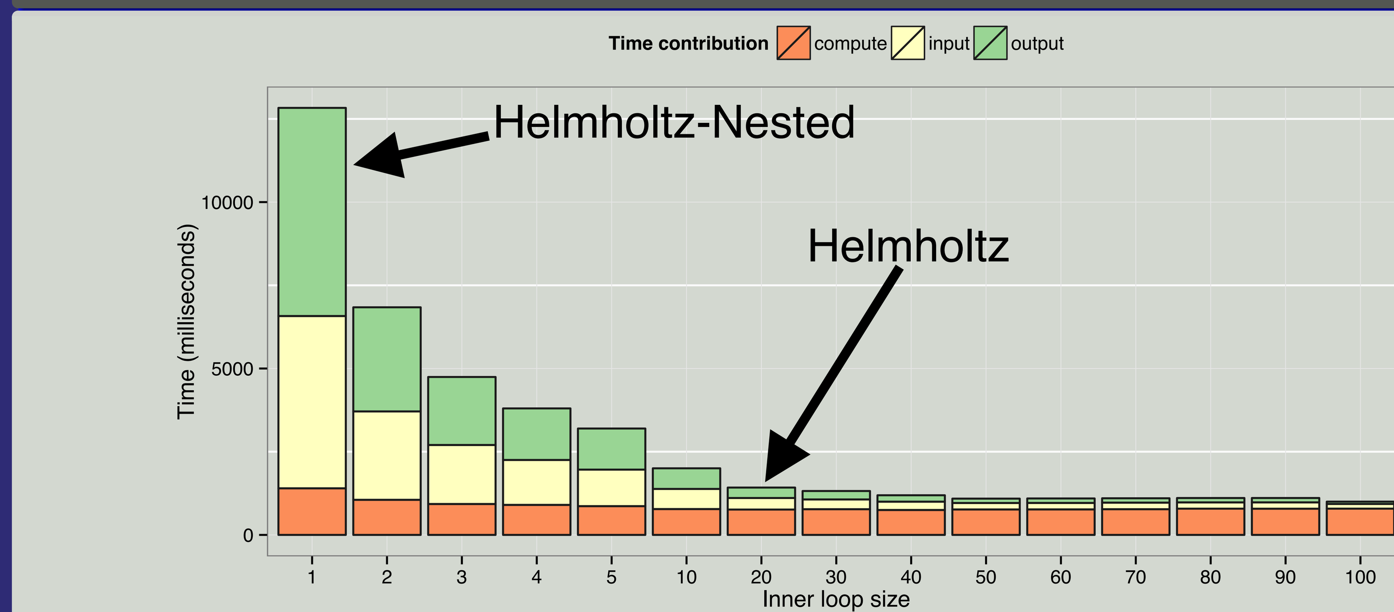
Memory Association Types



Results: Co-Scheduling Performance

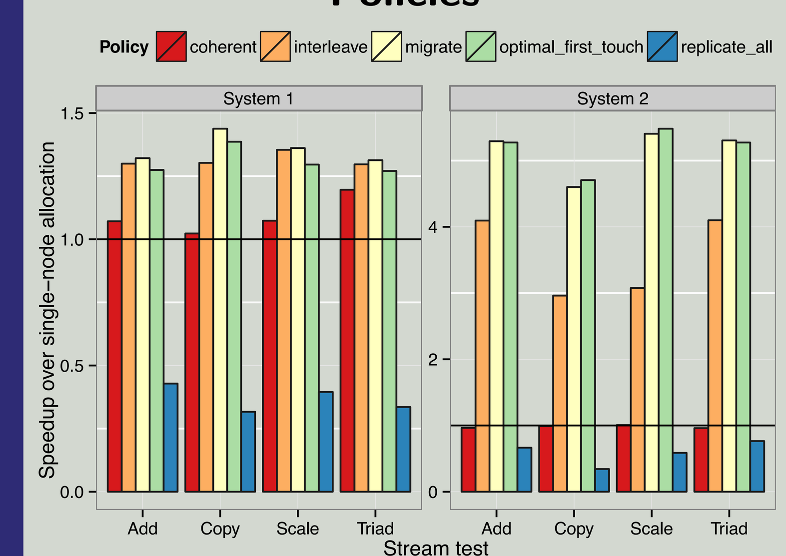


Results: Data-movement Cost of Frequent Re-Balancing



Results: Memory-Movement Optimization

Stream Bandwidth with NUMA Policies



GEMM CPU-only with NUMA Policies

Conclusions

- Partitioning simplifies a common pattern, while increasing the capabilities of the compiler and runtime
- 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
- When applied to mitigating NUMA affinity issues, we also see improvements of as much as 40% in the bandwidth of the stream benchmark, and greater than 3× performance improvement in the performance of dense matrix multiplication on the CPU with appropriate policies

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*, 2015, **under preparation**.

[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 **Accepted**.

[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.