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

Thomas R. W. Scogland  Wu-chun Feng
Department of Computer Science, Virginia Tech, Blacksburg, VA 24060 USA.

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

<table>
<thead>
<tr>
<th>Memory Node (MN) 0</th>
<th>GPU 0</th>
<th>GPU 1</th>
<th>GPU 2</th>
<th>GPU 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Node (MN) 1</td>
<td>12,407</td>
<td>8,794</td>
<td>3,851</td>
<td>3,855</td>
</tr>
</tbody>
</table>

Node, or Constellation?

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

How can we address hierarchical memory and multiple accelerators with a single, unified extension to OpenMP (or similar models)

Adaptation: Load-balancing for Partitioned Worksharing

Linear Program

\[ \sum_{j} t_j + t_{j+1} \]

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

2D Array:

CPU 1

Segmented Array:

12,16,5,15,10,7,8,11,4,14,12,6,10,3,9

CPU 2

Indexed Array:

CPU 6

Target array:

Target array:

Target array:

Results: Co Scheduling Performance

Results: Data-movement Cost of Frequent Re-Balancing

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 50x speedup over eight core CPU with four GPUs, and we show a nearly 2x 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 3x performance improvement in the performance of dense matrix multiplication on the CPU with appropriate policies

Related Papers


http://synerg.cs.vt.edu ~ trwscogland@cs.vt.edu