Compiler Based Workload Consolidation for Efficient Dynamic Parallelism on GPUs

Most irregular computations such as graph and tree algorithms can be expressed as patterns of irregular loops and parallel recursion. We propose a compiler framework to improve the efficiency of such irregular patterns written using Dynamic Parallelism by workload consolidation on GPUs.

Basic Template of DP Kernel

```
global void parent_kernel(){
  job = get_work_item();
  prework(job);
  if (condition) {
    [program dp consolidation grid buffer work](job);
    child_kernels[<block_dim, thread_dim>](job);
  } ...
  else work(job);
  postwork(thread_id);
}
```

Kernel After Consolidation

```
global void parent_kernel(){
  job = get_work_item();
  prework(job);
  if (condition) {
    [program dp consolidation grid buffer work](job);
    child_kernels[<block_dim, thread_dim>](job);
  } ...
  else work(job);
  postwork(thread_id);
}
```

STMD Patterns Optimization Variants

<table>
<thead>
<tr>
<th>Variant</th>
<th>Method to Track Traversal Order</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>REC</td>
<td>Stack of each thread</td>
<td>Naive parallelization</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Explicit user-defined stack</td>
<td>Recursive; Rope computed at runtime</td>
</tr>
<tr>
<td>Static</td>
<td>Static pointers installed on the tree</td>
<td>Iterative; Rope computed only once avoid rec/user-stack</td>
</tr>
</tbody>
</table>

GPU Specific Optimizations:
- Greed DR - Allows a thread to fetch a new query immediately upon finishing the current. Lockstep - Force threads in a warp to follow the same traversal path.

PTSD Pattern Optimization Variants

<table>
<thead>
<tr>
<th>Variant</th>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>REC</td>
<td>Parent kernels with thread creation and sync overheads</td>
<td>Reduced &amp; if REC calls buffering &amp; sync overhead</td>
</tr>
<tr>
<td>Flat</td>
<td>Flat Parallelism</td>
<td>Less work efficient &amp; Atomic</td>
</tr>
</tbody>
</table>

Microbenchmark and Results

1. for_loop(num_iterations) {
2.   ... 
3. } for_each_thread() {
4.   ... 
5. }

(a) Irregular memory accesses
6. if(threadId?b:d) { ... }
7. loop,(cost) 
8. else if (threadId>b:d, g threadId<d,b) { ... }
9. loop,(cost) 
10. ... 
11. ... 
12. else if (threadId>b:d, g threadId<d,b) { ... }
13. loop,(cost) 
14. (b) Control divergence
15. for_loop(num_iterations) {
16.   ...synrwo(){
17. } ... 
18. for_loop(num_iterations) {
19.   ...synrwo(){
20. } ... 
21. (c) Thread-block synchronization
22. for_loop(num_iterations) {
23.   if (threadId==b[d]a; g threadId<d[b]a) {
24.     shared_mem_write(sm_array, random_degree) {
25.     shared_mem_read(sm_array, random_degree) {
26.     ... 
27. } ... 
28. (d) Shared memory accesses

Results

Grid-level Consolidation is the best with correct kernel configuration, yielding 90x to 3300x speedup over basic DP-based solutions and 2x to 6x speedup over flat implementations.

Graph applications w/ iterative kernel launches can hardly benefit. Irregular computations such as BFS will suffer. Computations with regular memory accesses, few kernel invocations and no synchronization can run faster on hybrid architectures than on GPUs (NN).