Reduction Operations in Parallel Loops for GPGPUs

Rengan Xu, Xiaonan Tian, Yonghong Yan, Sunita Chandrasekaran and Barbara Chapman

Department of Computer Science
University of Houston

February 15, 2014
Outline

1. Motivation
2. Overview of OpenACC
3. Mapping Parallel Loops onto GPGPU Architecture
4. Parallelization of Reduction Operations for GPGPUs
5. Results
6. Conclusions
Motivation

- Reduction operations widely used in parallel loops
- Reduction impacts the performance of parallel loops significantly
- OpenACC reduction performance in commercial compilers varies and not fully supported
- No open-source implementation of OpenACC reduction
The standard of directive-based programming model for accelerator programming (GPU, APU, Xeon Phi, etc.)

Compute Directives
- parallel: more control by user
- kernels: more control by compiler

Three levels of parallelism
- gang: coarse-grained
- worker: fine-grained
- vector: vector parallelism
OpenUH - An Open Source OpenACC Compiler

OpenUH Compiler Infrastructure

- FRONTENDS (C, OpenACC)
- IPA (Inter Procedural Analyzer)
- PRELOWER (Preprocess OpenACC)
- WOPT (Global Scalar Optimizer)
- LOWER (Transformation of OpenACC)
- LNO (Loop Nest Optimizer)
- WHIRL2CUDA
- CG (Code for IA-32, IA-64, X86_64)

Source Code with OpenACC Directives

- GPU Code
  - NVCC Compiler
  - PTX Assembler
  - Loaded Dynamically

CPU Binary

- Linker
  - Runtime Library

Executable

Presented by Rengan Xu
Mapping Parallel Loops onto GPGPU Architecture

Figure: GPGPU Thread Block Hierarchy

- **gang**: map to thread block
- **worker**: Y-dimension of a thread block
- **vector**: X-dimension of a thread block
Reduction: Use binary operator to operate an input array and generate a single output value.

Reduction operator in OpenACC: $+$, $\ast$, $\&\&$, $||$, $\&$, $|$, $\wedge$, max, min

Associativity:
- $a_1 + a_2 + a_3$
- $(a_1 + a_2) + a_3$
- $a_1 + (a_2 + a_3)$

Commutativity:
- $a_1 + a_2 + a_3$
- $a_3 + a_1 + a_2$
- $a_2 + a_3 + a_1$
Reduction in Single-level Thread Parallelism

Reduction only in vector

```
#pragma acc parallel copyin(input) copyout(temp)
{
    #pragma acc loop gang
    for (k = 0; k < NK; k++) {
        #pragma acc loop worker
        for (j = 0; j < NJ; j++) {
            int i_sum = j;
            #pragma acc loop vector reduction(+:i_sum)
            for (i = 0; i < NI; i++)
                i_sum += input[k][j][i];
            temp[k][j][0] = i_sum;
        }
    }
}
```
Reduction in Single-level Thread Parallelism
Reduction only in vector

(a) Data and threads layout in global memory

(b) Data and threads layout in shared memory. Has bank conflict
Reduction in Single-level Thread Parallelism

Reduction only in vector (OpenUH way)

(c) Data and threads layout in global memory

(d) Data and threads layout in shared memory. NO bank conflict
Reduction in Single-level Thread Parallelism

Reduction only in vector

Figure: Interleaved Log-step Reduction Algorithm

Optimizations:
- Sequential addressing
- Loop unrolling
- Algorithm cascading
Reduction in Single-level Thread Parallelism

Reduction only in worker

```c
#pragma acc parallel copyin(input) copyout(temp)
{
    #pragma acc loop gang
    for (k = 0; k < NK; k++) {
        int j_sum = k;
        #pragma acc loop worker reduction(+:j_sum)
        for (j = 0; j < NJ; j++) {
            #pragma acc loop vector
            for (i = 0; i < NI; i++)
                temp[k][j][i] = input[k][j][i];
            j_sum += temp[k][j][0];
        }
        temp[k][0][0] = j_sum;
    }
}
```
Reduction in Single-level Thread Parallelism

Reduction only in worker

(a) Data and threads layout in global memory

(b) Data and threads layout in shared memory. Has bank conflict, more shared memory.
Reduction in Single-level Thread Parallelism

Reduction only in worker (OpenUH way)

(c) Data and threads layout in global memory

(d) Data and threads layout in shared memory. NO bank conflict, less shared memory.
Reduction in Single-level Thread Parallelism

Reduction only in gang

- No built-in synchronization mechanisms to synchronize all blocks/gangs
- Create a temporary buffer in global memory with the size equal to the number of blocks/gangs
- Populate the buffer by all blocks/gangs
- Launch another kernel to do the vector reduction within only one block
Parallelization of Reduction Operations for GPGPUs

Reduction across Multi-level Thread Parallelism (RMP)

RMP in different loops

```c
#pragma acc parallel copyin(input) copyout(temp)
{
    #pragma acc loop gang
    for (k = 0; k < NK; k++) {
        int j_sum = k;
        #pragma acc loop worker reduction(+:j_sum)
        for (j = 0; j < NJ; j++) {
            #pragma acc loop vector reduction(+:j_sum)
            for (i = 0; i < NI; i++)
                j_sum += input[k][j][i];
        } j_sum += input[k][j][i];
        temp[k] = j_sum;
    }
}
```
Reduction across Multi-level Thread Parallelism (RMP)

RMP in different loops

- CAPS compiler adds reduction clause to both j and i loops
- OpenUH compiler adds reduction clause to only one place
- The reduction is done ONCE by all threads in one block
- Alternative way: vector reduction first, then worker reduction
- Other possible combinations
  - gang worker
  - gang worker vector
- If gang involved, use global memory and launch another reduction kernel
Reduction across Multi-level Thread Parallelism (RMP)

RMP in the same loop

```c
sum = 0;
#pragma acc parallel copyin(input)
{
    #pragma acc loop gang worker vector reduction(+:sum)
    for (i = 0; i < NI; i++)
    sum += input[i];
}
```

- Create a buffer with size of all threads doing reduction
- First perform partial reduction, then launch another kernel
- Shared or global memory? - decided by gang
Results

Experimental Platform

- Host: 24 Intel Xeon x86_64 cores, 32GB memory
- Device: NVIDIA Kepler GPU (k20c), 5GB global memory
- Software: CAPS 3.4.0, PGI 3.10, OpenUH, CUDA 5.5, GCC 4.4.7
- Reduction data size: 1M
- Evaluation benchmarks:
  - Self-written reduction testsuite
  - 2D Heat Equation
  - Matrix Multiplication
  - Monte Carlo PI
# Results

Result of reduction testsuite

F: test failed. CE: compilation error

<table>
<thead>
<tr>
<th>Reduction Position</th>
<th>Reduction Operator</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>OpenUH</td>
</tr>
<tr>
<td>gang</td>
<td>+</td>
<td>151.27</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>156.01</td>
</tr>
<tr>
<td>worker</td>
<td>+</td>
<td>399.25</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>413.35</td>
</tr>
<tr>
<td>vector</td>
<td>+</td>
<td>268.47</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>269.80</td>
</tr>
<tr>
<td>gang worker</td>
<td>+</td>
<td>107.49</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>113.36</td>
</tr>
<tr>
<td>worker vector</td>
<td>+</td>
<td>50.58</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>51.20</td>
</tr>
<tr>
<td>gang worker vector</td>
<td>+</td>
<td>8.77</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>8.15</td>
</tr>
<tr>
<td>same line gang worker vector</td>
<td>+</td>
<td>7.55</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>7.25</td>
</tr>
</tbody>
</table>
Results

Performance comparison

(e) Int [+]

(f) Float [+]

(g) Double [+]

Presented by Rengan Xu
Results

Performance comparison

(h) Int [*]

(i) Float [*]

(j) Double [*]
Results

Performance comparison

(k) 2D Heat Equation

[\text{[max]}]

(l) Matrix Multiplication

[\text{[+]}]

(m) Monte Carlo PI

[\text{[+]}]
Conclusions

- Present all possible reduction cases in OpenACC
- Demonstrate efficient reduction parallelization strategies in an open-source OpenACC compiler - OpenUH
- OpenUH performance is competitive to commercial compilers
- Similar strategies can be applied to OpenMP 4.0, ignore worker
- OpenUH: http://web.cs.uh.edu/~openuh/