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

August 30, 2014

Outline

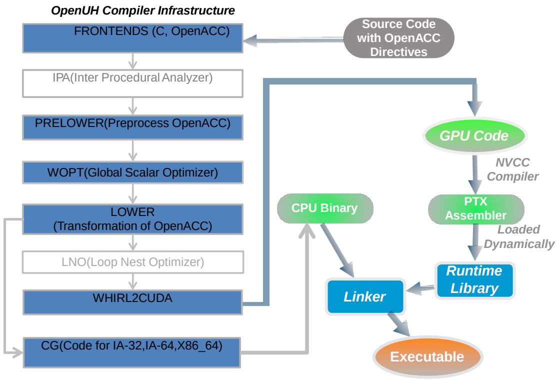
Motivation

- 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

Overview of OpenACC

- 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
Runtime Library
CPU Binary
Linker
Executable

Mapping Parallel Loops onto GPGPU Architecture

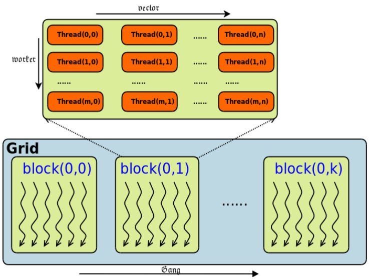


Figure : GPGPU Thread Block Hierarchy

- gang: map to thread block
- worker: Y-dimension of a thread block
- vector: X-dimension of a thread block

Parallelization of Reduction Operations for GPGPUs

Reduction Properties

- Reduction: Use binary operator to operate an input array and generate a single output value
- Reduction operator in OpenACC: **+, *, &&, ||, &, |, ^, max, min**
- Associativity:
  - a1 + a2 + a3
  - (a1 + a2) + a3
  - a1 + (a2 + a3)
- Commutativity:
  - a1 + a2 + a3
  - a3 + a1 + a2
  - a2 + a3 + a1

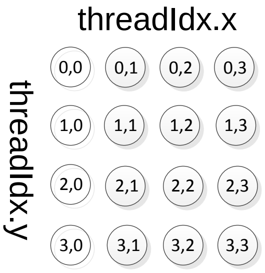
Reduction in Single-level Thread Parallelism

Reduction only in vector

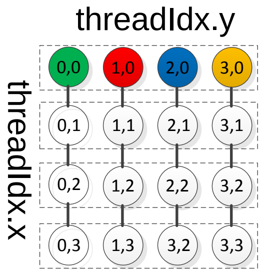
```
#pragma acc parallel copyin(input) copyout(temp)
{
  #pragma acc loop gang
  for(k=0; k<NK; k++){
    #pragma acc loop worker
    for(j=0; j<NJ; j++){
      int i_sum = j;
      #pragma acc loop vector reduction(+:i_sum)
      for(i=0; i<NI; i++)
         i_sum += input[k][j][i];
      temp[k][j][0]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

Reduction in Single-level Thread Parallelism

Reduction only in vector



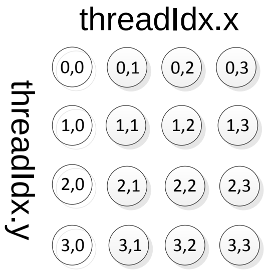
(a) Data and threads layout in global memory



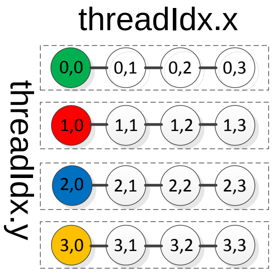
(b) Data and threads layout in shared memory. Has bank conflict

Reduction in Single-level Thread Parallelism

Reduction only in vector (OpenUH way)



(a) Data and threads layout in global memory



(b) Data and threads layout in shared memory. NO bank conflict

Reduction in Single-level Thread Parallelism

Reduction only in vector

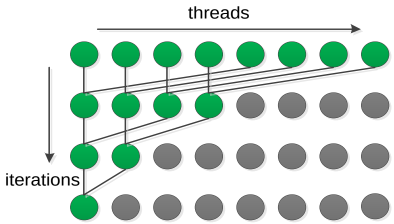


Figure : Interleaved Log-step Reduction Algorithm

Optimizations:

- Sequential addressing
- Loop unrolling
- Algorithm cascading

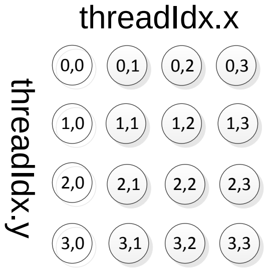
Reduction in Single-level Thread Parallelism

Reduction only in worker

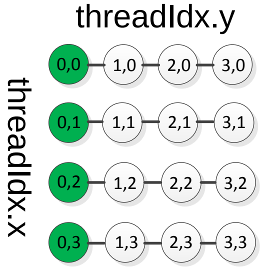
```
#pragma acc parallel copyin(input) copyout(temp)
{
  #pragma acc loop gang
  for(k=0; k<NK; k++){
    int j_sum = k;
    #pragma acc loop worker reduction(+:j_sum)
    for(j=0; j<NJ; j++){
      #pragma acc loop vector
      for(i=0; i<NI; i++)
        temp[k][j][i] = input[k][j][i];
      j_sum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

Reduction in Single-level Thread Parallelism

Reduction only in worker



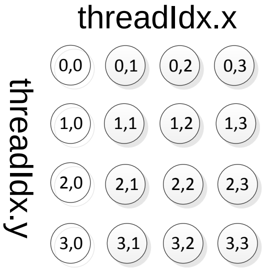
(a) Data and threads layout in global memory



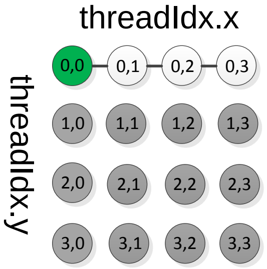
(b) Data and threads layout in shared memory. Has bank conflict, more shared memory.

Reduction in Single-level Thread Parallelism

Reduction only in worker (OpenUH way)



(a) Data and threads layout in global memory



(b) 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

Reduction across Multi-level Thread Parallelism (RMP)

RMP in different loops

```
#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];
    }
    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

```
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

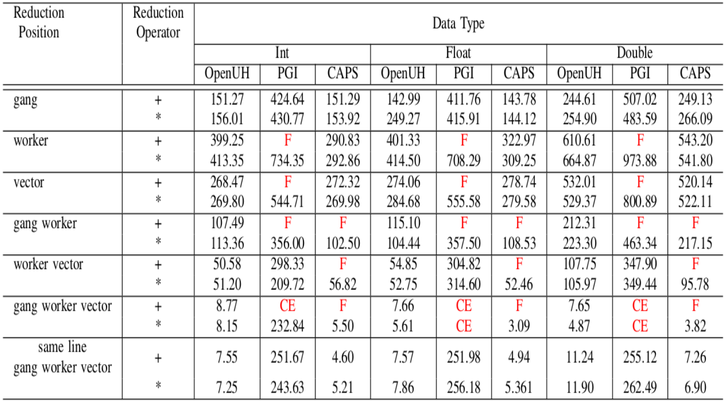
Experimental Platform

- 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

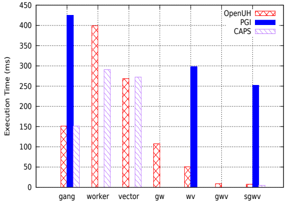
F: test failed. CE: compilation error



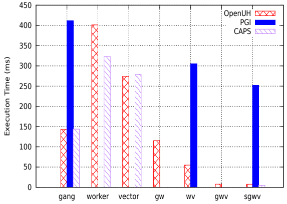
Reduction Position	Reduction Operator	Data Type								
		Int			Float			Double		
		OpenUH	PGI	CAPS	OpenUH	PGI	CAPS	OpenUH	PGI	CAPS
gang	+	151.27	424.64	151.29	142.99	411.76	143.78	244.61	507.02	249.13
	*	156.01	430.77	153.92	249.27	415.91	144.12	254.90	483.59	266.09
worker	+	399.25	F	290.83	401.33	F	322.97	610.61	F	543.20
	*	413.35	734.35	292.86	414.50	708.29	309.25	664.87	973.88	541.80
vector	+	268.47	F	272.32	274.06	F	278.74	532.01	F	520.14
	*	269.80	544.71	269.98	284.68	555.58	279.58	529.37	800.89	522.11
gang worker	+	107.49	F	F	115.10	F	F	212.31	F	F
	*	113.36	356.00	102.50	104.44	357.50	108.53	223.30	463.34	217.15
worker vector	+	50.58	298.33	F	54.85	304.82	F	107.75	347.90	F
	*	51.20	209.72	56.82	52.75	314.60	52.46	105.97	349.44	95.78
gang worker vector	+	8.77	CE	F	7.66	CE	F	7.65	CE	F
	*	8.15	232.84	5.50	5.61	CE	3.09	4.87	CE	3.82
same line gang worker vector	+	7.55	251.67	4.60	7.57	251.98	4.94	11.24	255.12	7.26
	*	7.25	243.63	5.21	7.86	256.18	5.361	11.90	262.49	6.90

Results

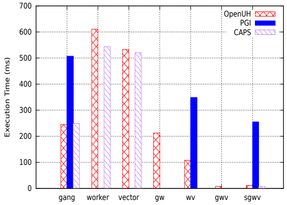
Performance comparison



(a) Int [+]



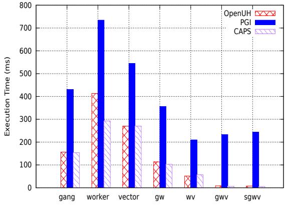
(b) Float [+]



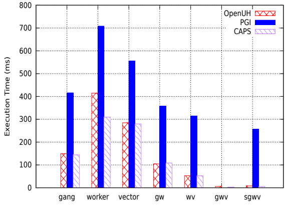
(c) Double [+]

Results

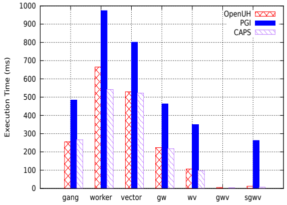
Performance comparison



(a) Int [*]



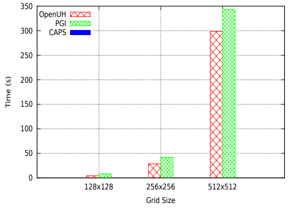
(b) Float [*]



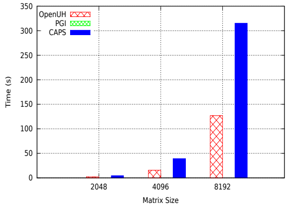
(c) Double [*]

Results

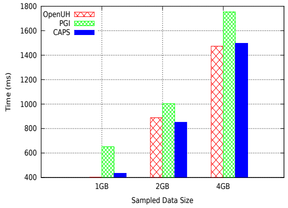
Performance comparison



(a) 2D Heat Equation [max]



(b) Matrix Multiplication [+]



(c) Monte Carlo PI [+]

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/