Compiling a High-level Directive-Based Programming Model for GPGPUs

Xiaonan Tian Rengan Xu Yonghong Yan
Zhifeng Yun Sunita Chandrasekaran Barbara Chapman

Department of Computer Science
University of Houston

LCPC Workshop 2013

Outline

- OpenACC 1.0 was announced on SC'11
- Goals: improve performance, productivity and portability

```
#pragma acc data copyin( a[0:n], b[0:n] ), copyout( c[0:n] )
{
   #pragma acc kernels
   {
      #pragma acc loop independent
      for ( i =  0;  i < n; i++ ) {
         c[i] = a[i] + b[i];
      }
   }
}
```

Motivation and Contribution

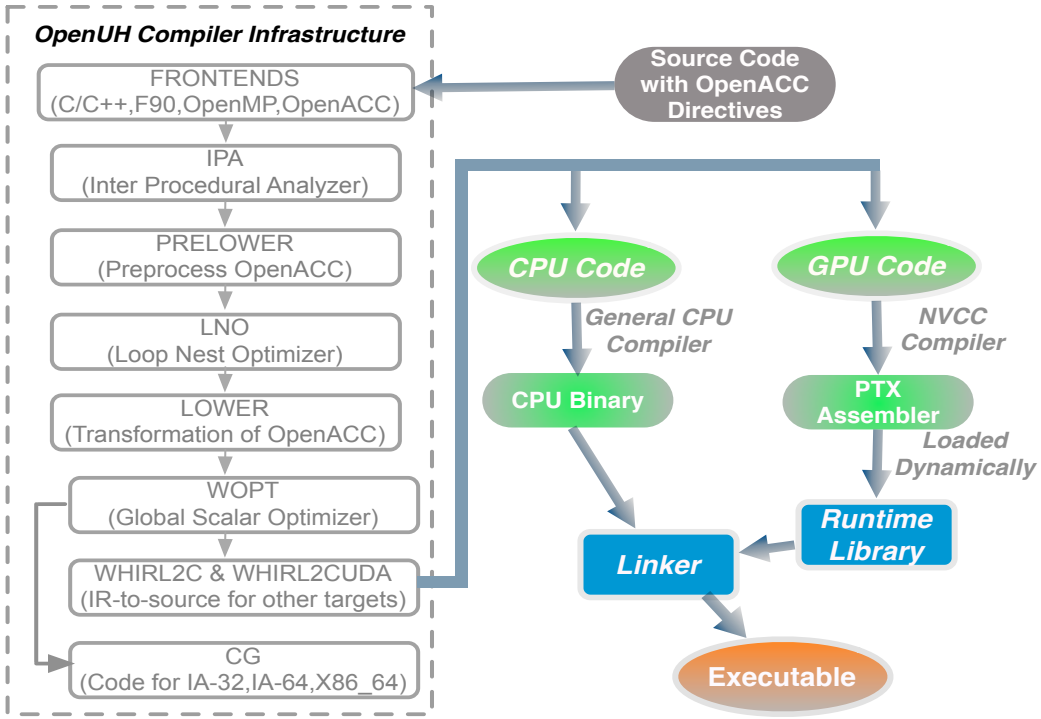
Motivation

- The community needs opensource openacc compiler for research
- How do sequential loops map to parallel loops in GPGPUs?

Contribution

- Constructed a prototype open-source OpenACC compiler
- Propose multiple loop mapping algorithms
  - Efficiently distribute loop iterations
  - Cover single, double and triple nested loops
- OpenUH compiler adopts a source-to-source approach
  - readable CUDA source code for GPGPUs

- OpenUH: a branch of the open source Open64 compiler.
- Source-to-Source strategy.

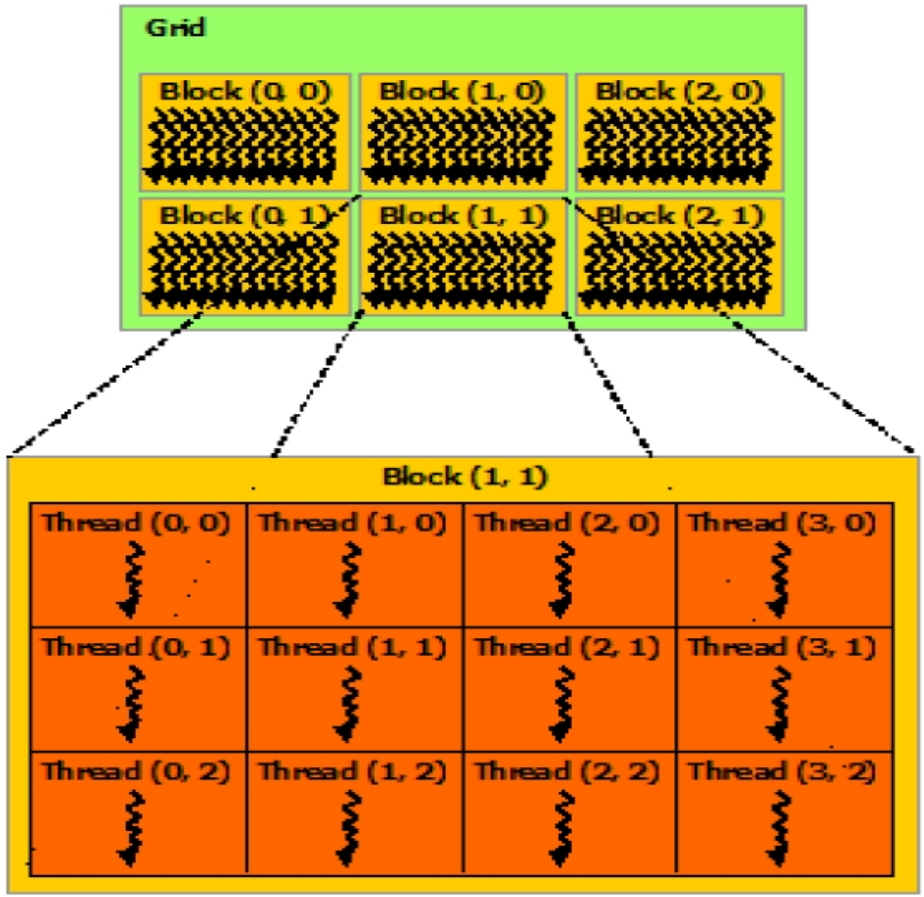


Massively Parallel

- Major challenge
  - How to effectively map the loop nest to the GPU parallel system?
- OpenACC provides three levels of parallelism:
  - Coarse-grain parallelism **gang**
  - Fine-grain parallelism **worker**
  - Vector parallism **vector**
- Nvidia GPGPUs have two levels of parallelism:
  - Block-level
  - Thread-level

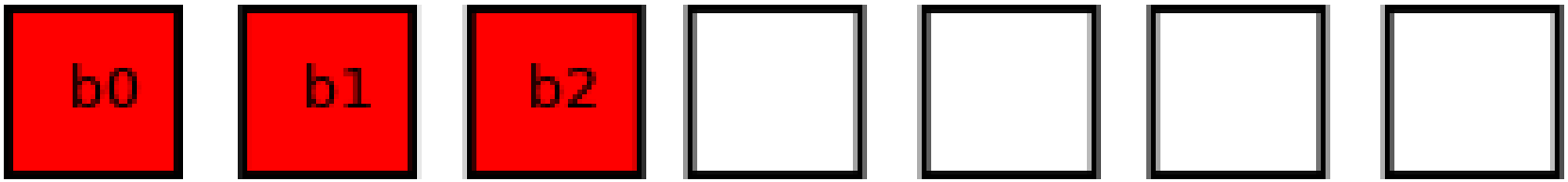
NVIDIA GPGPU Architecture Overview

- Thread blocks are organized into a 1D, 2D, or 3D grid
- Threads form a 1D, 2D, or 3D thread block.



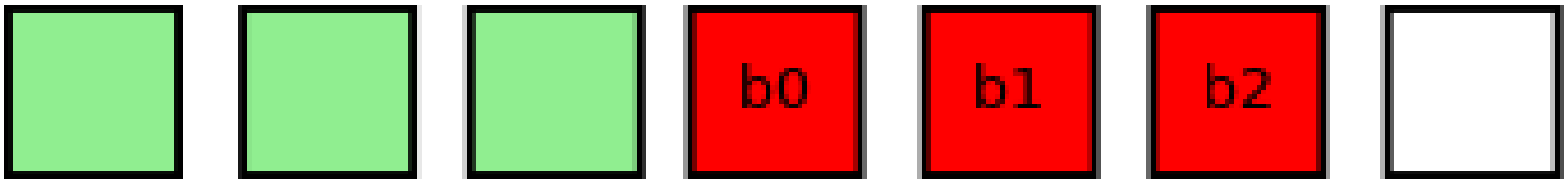
Basic Loop Mapping 1

```
#pragma acc loop gang(3)
for ( i = x1; i < X1; i++ ) {
......
}
```



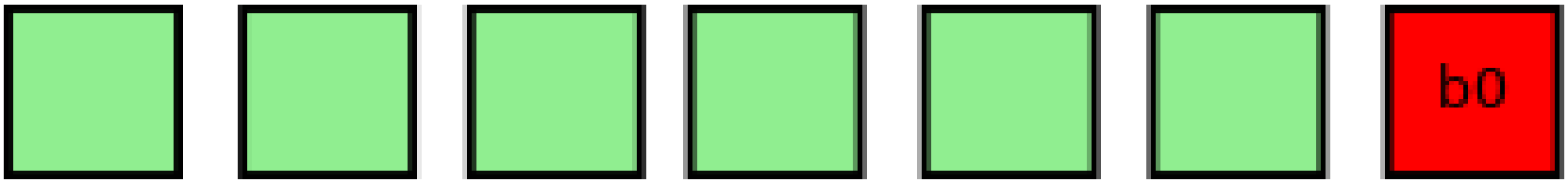
Basic Loop Mapping 1

```
#pragma acc loop gang(3)
for ( i = x1; i < X1; i++ ) {
......
}
```



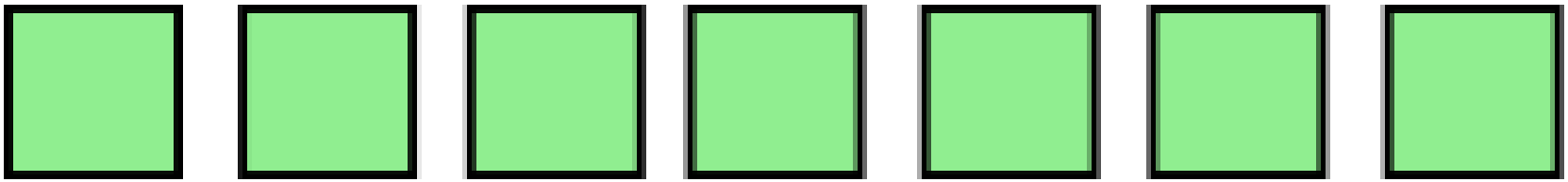
Basic Loop Mapping 1

```
#pragma acc loop gang(3)
for ( i = x1; i < X1; i++ ) {
......
}
```



Basic Loop Mapping 1

```
#pragma acc loop gang(3)
for ( i = x1; i < X1; i++ ) {
......
}
```

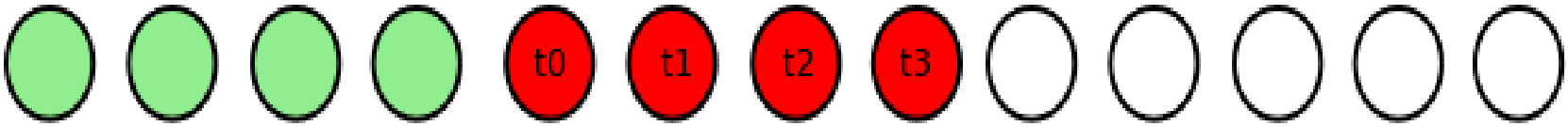


Basic Loop Mapping 2

```
#pragma acc loop vector(4)
for ( i = x1; i < X1; i++ ) {
......
}
```

Basic Loop Mapping 2

```
#pragma acc loop vector(4)
for ( i = x1; i < X1; i++ ) {
......
}
```

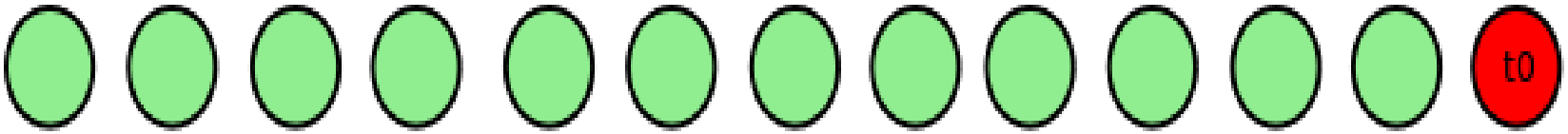


Basic Loop Mapping 2

```
#pragma acc loop vector(4)
for ( i = x1; i < X1; i++ ) {
......
}
```

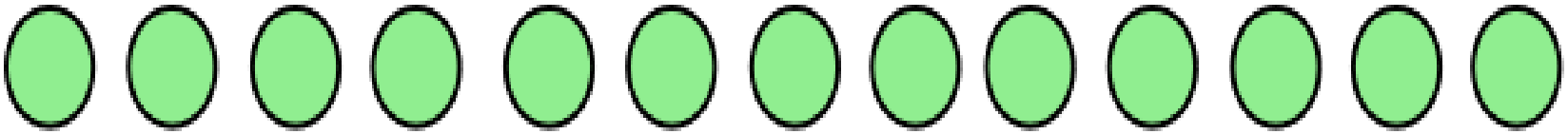
Basic Loop Mapping 2

```
#pragma acc loop vector(4)
for ( i = x1; i < X1; i++ ) {
......
}
```



Basic Loop Mapping 2

```
#pragma acc loop vector(4)
for ( i = x1; i < X1; i++ ) {
......
}
```

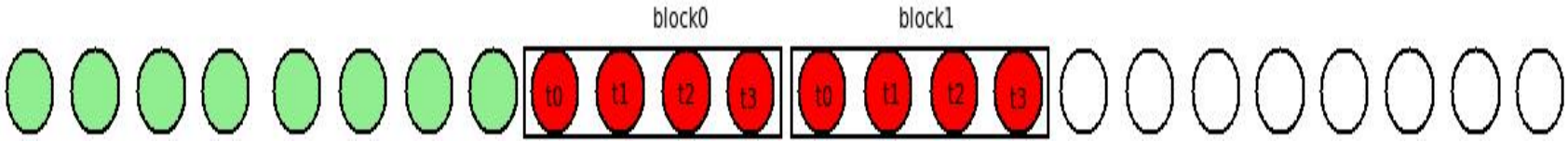


Basic Loop Mapping 3

```
#pragma acc loop gang(2) vector(4)
for ( i = x1; i < X1; i++ ) {
......
}
```

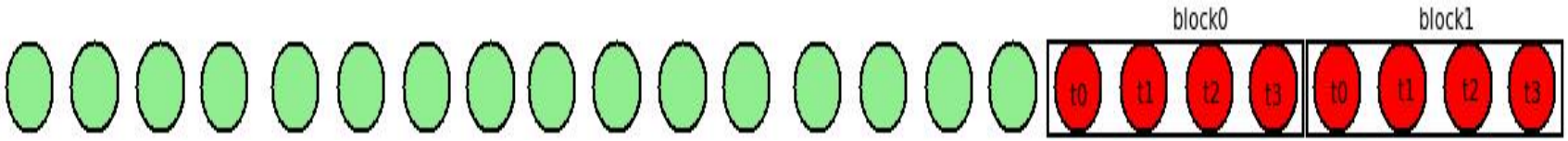
Basic Loop Mapping 3

```
#pragma acc loop gang(2) vector(4)
for ( i = x1; i < X1; i++ ) {
......
}
```



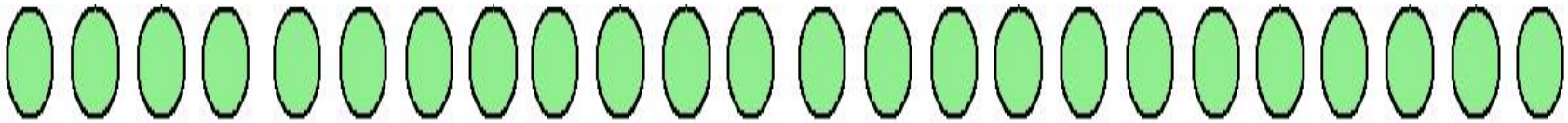
Basic Loop Mapping 3

```
#pragma acc loop gang(2) vector(4)
for ( i = x1; i < X1; i++ ) {
......
}
```



Basic Loop Mapping 3

```
#pragma acc loop gang(2) vector(4)
for ( i = x1; i < X1; i++ ) {
......
}
```



Double Nested Loop Mapping Algorithms Scenerio 1

```
#pragma acc loop gang
for ( i = x1;  i < X1; i++ ) {
   #pragma acc loop vector
   for ( j = y1;  j < Y1; j++ ) {
     ......
   }
}
```

Figure: Map-g-v

```
#pragma acc loop gang vector
for ( i = x1;  i < X1; i++ ) {
   #pragma acc loop vector
   for ( j = y1;  j < Y1; j++ ) {
     ......
   }
}
```

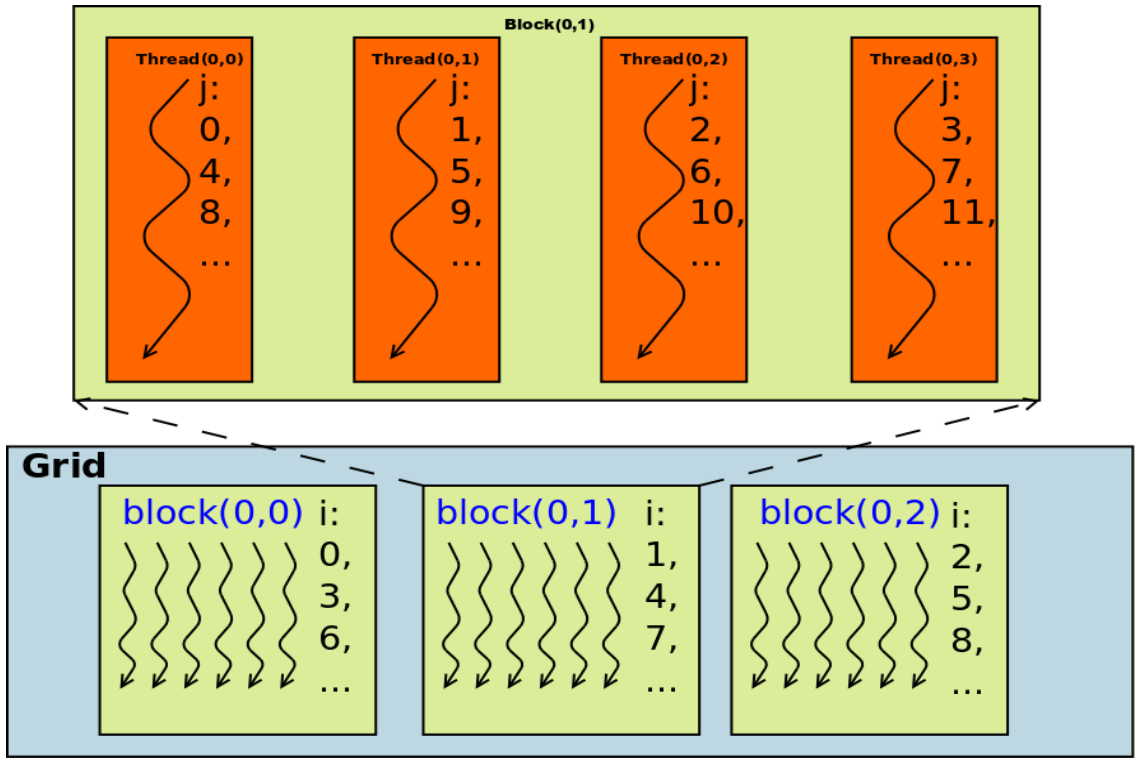
Figure: Map-gv-v

Table: Double Nested Loop Mapping in OpenUH

Double Loop	**Map-g-v**	**Map-gv-v**
Outer loop	gang $\rightarrow$ block.x	gang vector $\rightarrow$ block.x thread.y
Inner loop	vector $\rightarrow$ thread.x	vector $\rightarrow$ thread.x

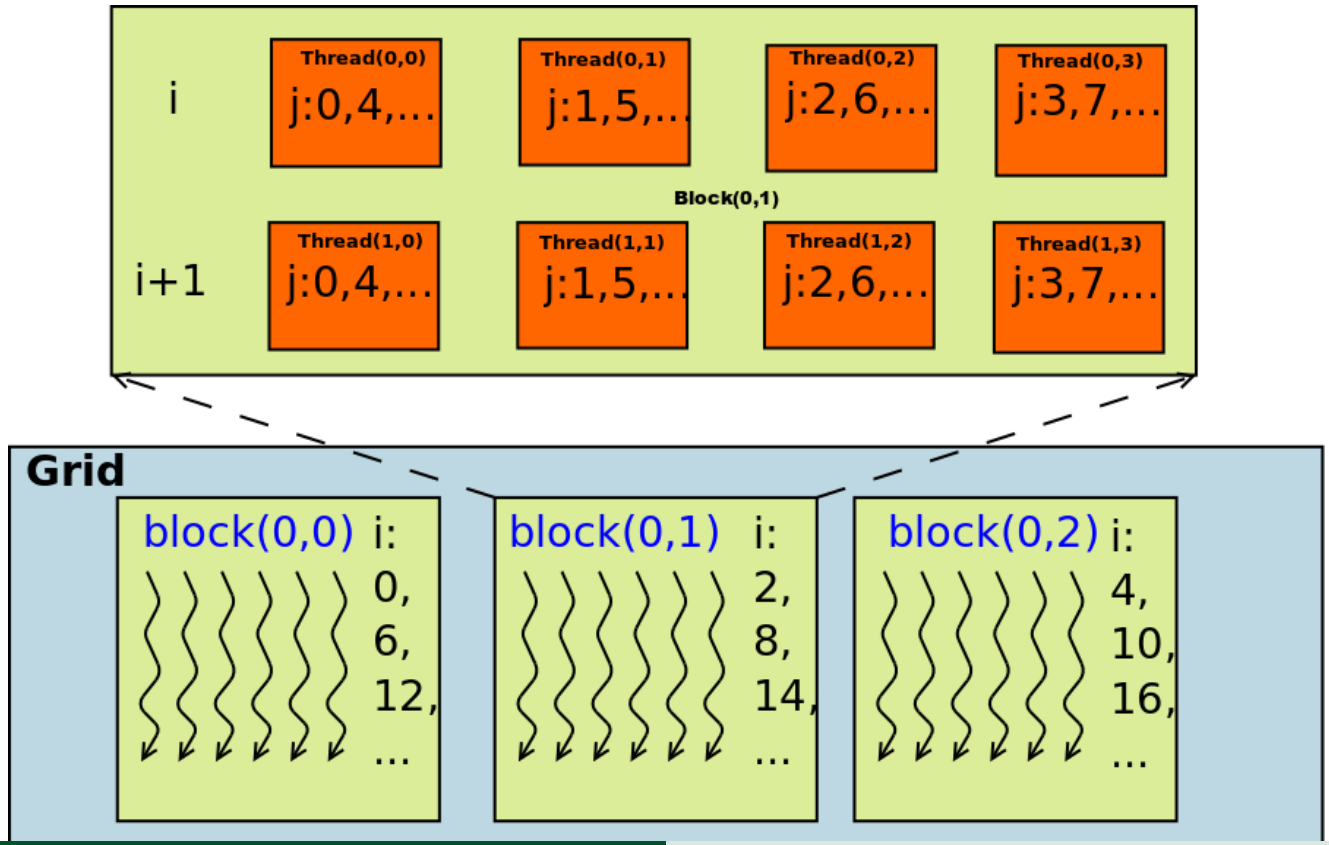
Map-g-v in GPU Topology

```
#pragma acc loop gang(3)
for ( i = x1; i < X1; i++ ) {
#pragma acc loop  vector(4)
for ( j = y1; j < Y1; j++ ) {..... }
}
```



Map-gv-v in GPU Topology

```
#pragma acc loop gang(3) vector(2)
for ( i = x1; i < X1; i++ ) {
#pragma acc loop  vector(4)
for ( j = y1; j < Y1; j++ ) {..... }
}
```



Double Nested Loop Mapping Algorithms Scenerio 2

```
#pragma acc loop gang
for ( i = x1;  i < X1; i++ ) {
   #pragma acc loop gang vector
   for ( j = y1;  j < Y1; j++ ) {
     ......
   }
}
```

Figure: Map-g-gv

```
#pragma acc loop gang vector
for ( i = x1;  i < X1; i++ ) {
   #pragma acc loop gang vector
   for ( j = y1;  j < Y1; j++ ) {
     ......
   }
}
```

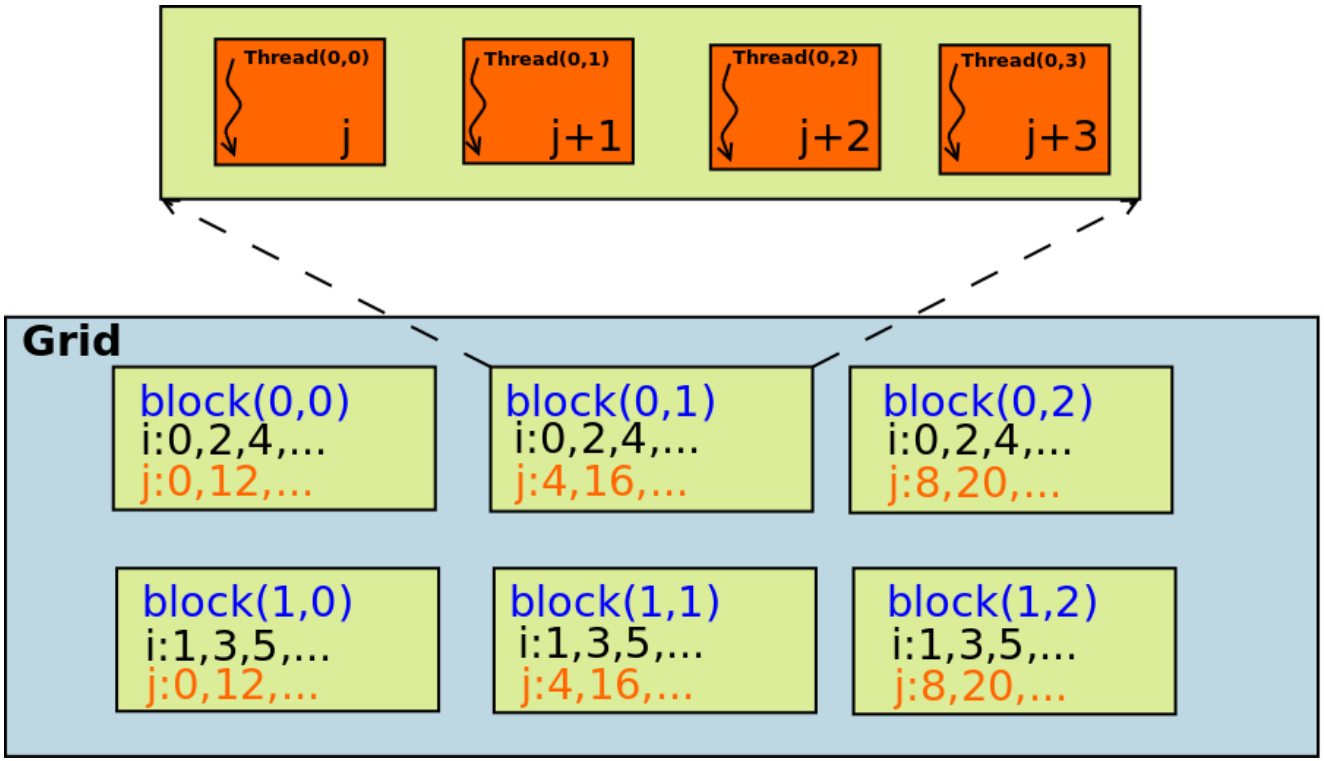
Figure: Map-gv-gv

Table: Double Nested Loop Mapping in OpenUH

Double Loop	**Map-g-gv**	**Map-gv-gv**
Outer loop	gang → block.y	gang vector → block.y thread.y
Inner loop	gang vector → block.x thread.x	gang vector → block.x thread.x

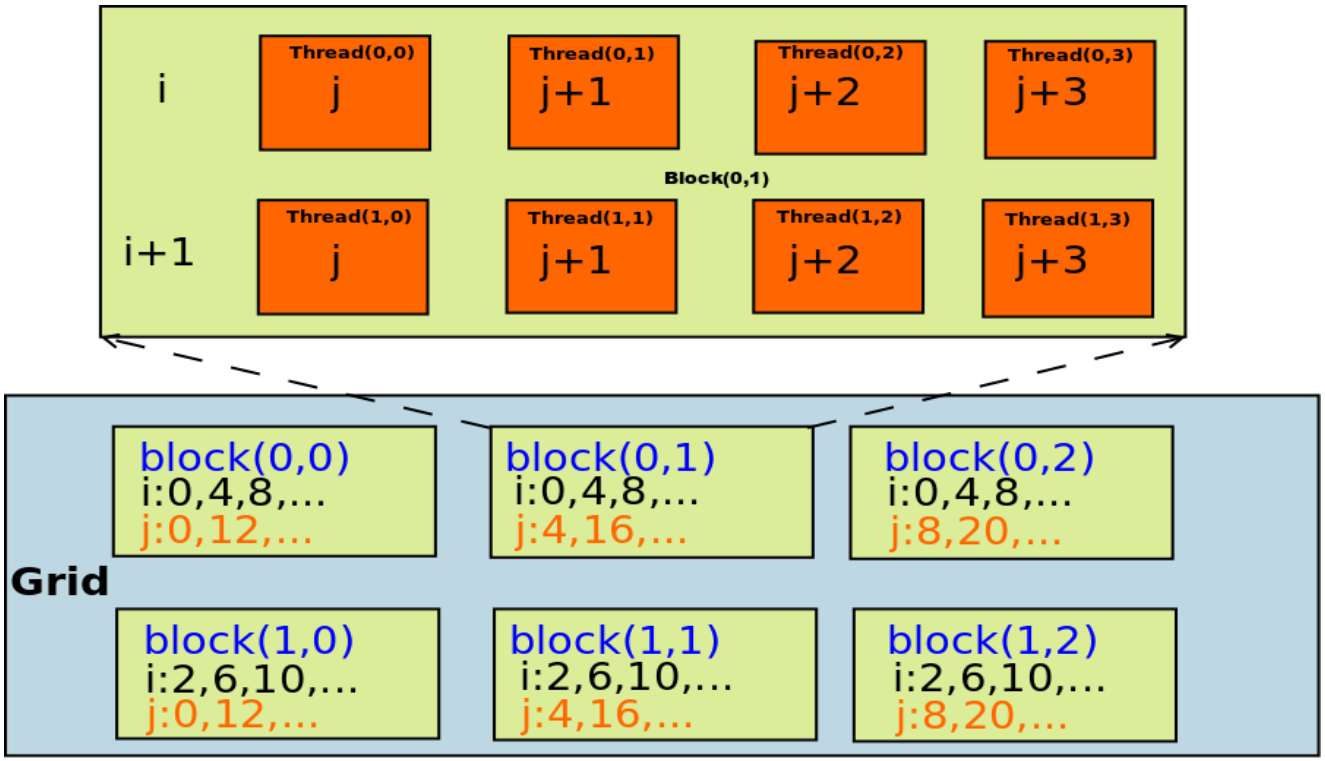
Map-g-gv in GPU Topology

```
#pragma acc loop gang(2)
for ( i = x1; i < X1; i++ ) {
#pragma acc loop  gang(3) vector(4)
for ( j = y1; j < Y1; j++ ) {..... }
}
```



Map-gv-gv in GPU Topology

```
#pragma acc loop gang(2) vector(2)
for ( i = x1; i < X1; i++ ) {
#pragma acc loop  gang(3) vector(4)
for ( j = y1; j < Y1; j++ ) {..... }
}
```



Triple Nested Loop Mapping Algorithms1

```
#pragma acc loop gang
for ( i = x1;  i < X1; i++ ) {
   #pragma acc loop gang vector
   for ( j = y1;  j < Y1; j++ ) {
      #pragma acc loop vector
      for ( k =z1; k < Z1; k++ ) {
         ......
      }
   }
}
```

Figure: Map3-1

Table: Triple Nested Loop Mapping in OpenUH-1

Triple Loop	Map3_1
Outermost Loop	gang $\rightarrow$ block.x
Middle Loop	gang vector $\rightarrow$ block.y thread.y
Innermost Loop	vector $\rightarrow$ thread.x

Triple Nested Loop Mapping Algorithms2

```
#pragma acc loop vector
for ( i = x1;  i < X1; i++ ) {
   #pragma acc loop gang vector
   for ( j = y1;  j < Y1; j++ ) {
      #pragma acc loop gang vector
      for ( k =z1; k < Z1; k++ ) {
         ......
      }
   }
}
```

Figure: Map3-2

Table: Triple Nested Loop Mapping in OpenUH-1

Triple Loop	Map3_2
Outermost Loop	vector → thread.z
Middle Loop	gang vector → block.y thread.y
Innermost Loop	gang vector → block.x thread.x

Triple Nested Loop Mapping Algorithms3

```
#pragma acc loop vector
for ( i = x1;  i < X1; i++ ) {
   #pragma acc loop gang vector
   for ( j = y1;  j < Y1; j++ ) {
      #pragma acc loop gang
      for ( k =z1; k < Z1; k++ ) {
         ......
      }
   }
}
```

Figure: Map3-3

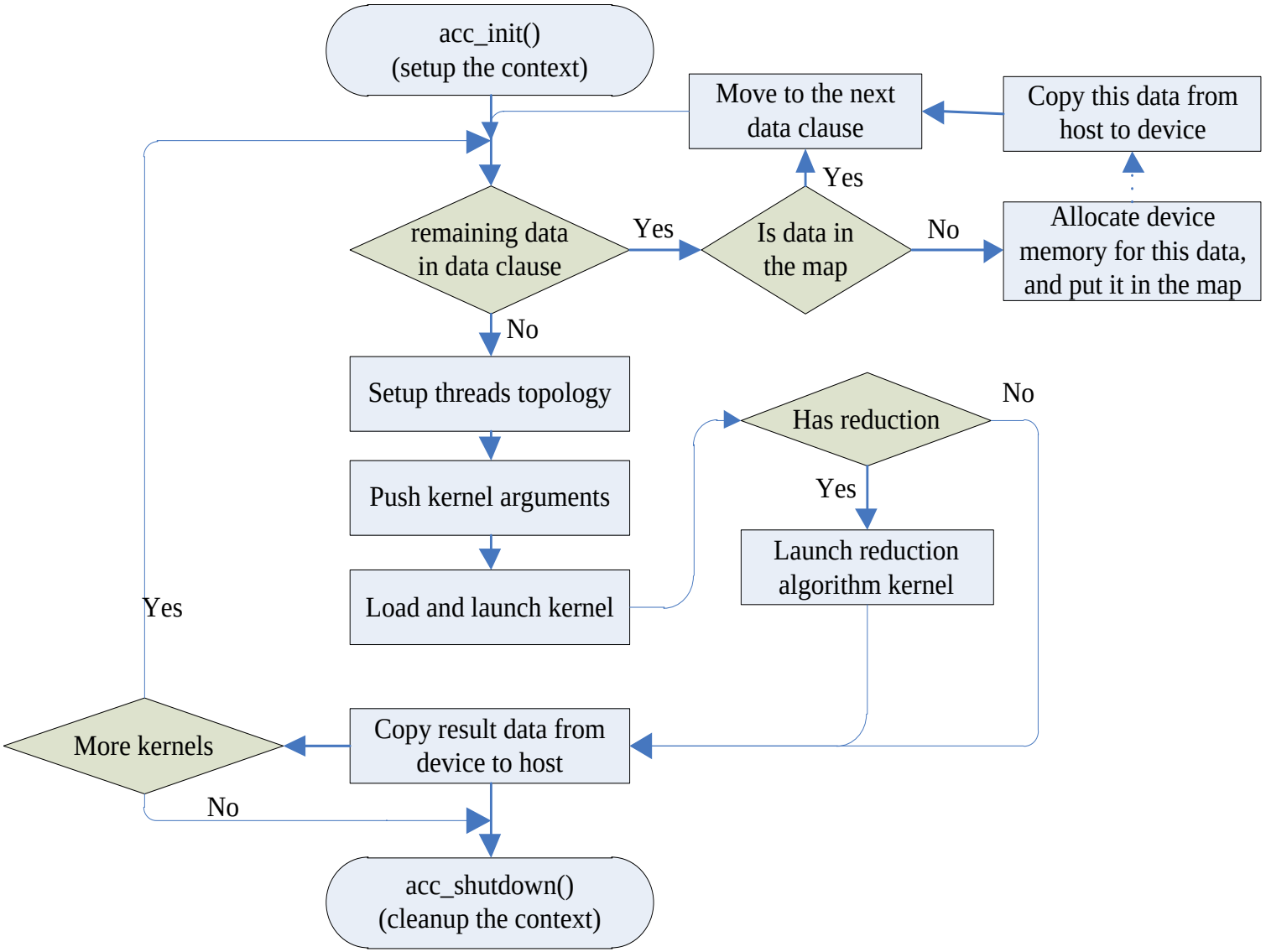
Table: Triple Nested Loop Mapping in OpenUH-2

Triple Loop	Map3_3
Outermost Loop	vector → thread.x
Middle Loop	gang vector → block.y thread.y
Innermost Loop	gang → block.x

Runtime Components

- Context Module
  - Create and manage the virtual execution environment
  - e.g. **acc_init()** and **acc_shutdown()**
- Memory Manager
  - Manage the data movement between the host and device
  - Runtime routines for **data** and **update** directives
  - Maintain a global hash map to implement **present** clause
- Kernel Loader
  - Load the generated cuda kernel file
  - Setup the threads topology
  - Push all arguments into the kernel parameter stack space
  - Launch the specified kernel

Runtime Execution Flow



Experiment Setup

- OpenUH is evaluated using Performance Test Suite, Stencil benchmark and DGEMM
- Machine: Intel Xeon x86_64 CPU with 32GB main memory, and a NVIDIA Kepler GPU card (K20)
- OpenUH: GCC 4.4.7 (O0) and nvcc 5.0.
- Double nested loopnest and triple nested loopnest are evaluated

Double Nested Loop Mapping Algorithms Experiment

GPU gang and vector configurations for each benchmark

Table: Threads used in double loop mappings

Benchmark	Double Loop	Map-g-v	Map-gv-v	Map-g-gv	Map-gv-gv
Jacobi	Outer loop	2048	1024x2	2046	1023x2
(2048x2048)	Inner loop	128	128	16x128	16x128
DGEMM	Outer loop	8192	4096x2	8192	4096x2
(8192x8192)	Inner loop	128	128	64x128	64x128
Gaussblur	Outer loop	1024	512x2	1020	510x2
(1024x1024)	Inner loop	128	128	8x128	8x128

Triple Nested Loop Mapping Algorithms Experiment

GPU gang and vector configurations for each benchmark

Table: Threads used in triple loop mappings

Benchmark	Triple Loop	Map-g-gv-v	Map-v-gv-gv	Map-v-gv-g
Stencil (512x512x64)	outermost loop	510	2	128
	middle loop	255x2	255x4	255x2
	innermost loop	128	16x64	62
Laplacian (128x128x128)	outermost loop	63	2	128
	middle loop	126x2	32x4	2
	innermost loop	128	2x64	126
Wave13pt (128x128x128)	outermost loop	64	2	128
	middle loop	124x2	31x4	2
	innermost loop	128	2x64	124

Double and Triple Nested Loop Mapping Algorithms Results

OpenUH Compiler Performance

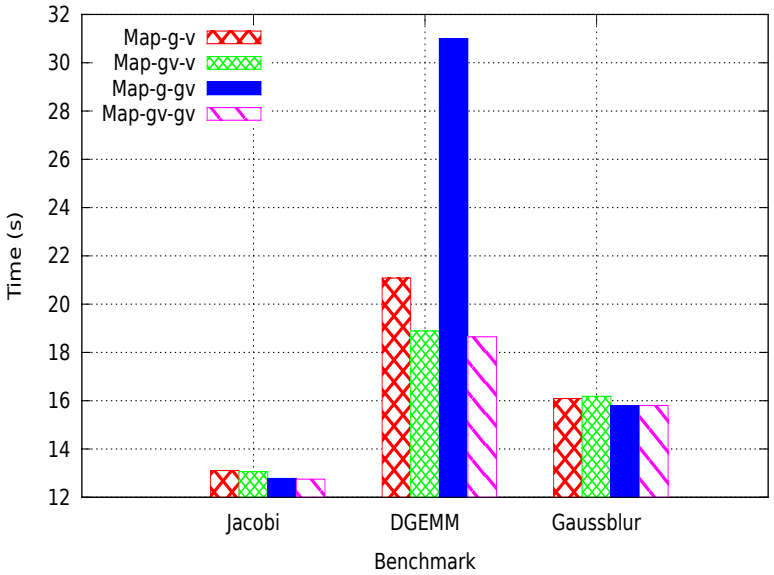


Figure: Performance of Double nested loop mapping

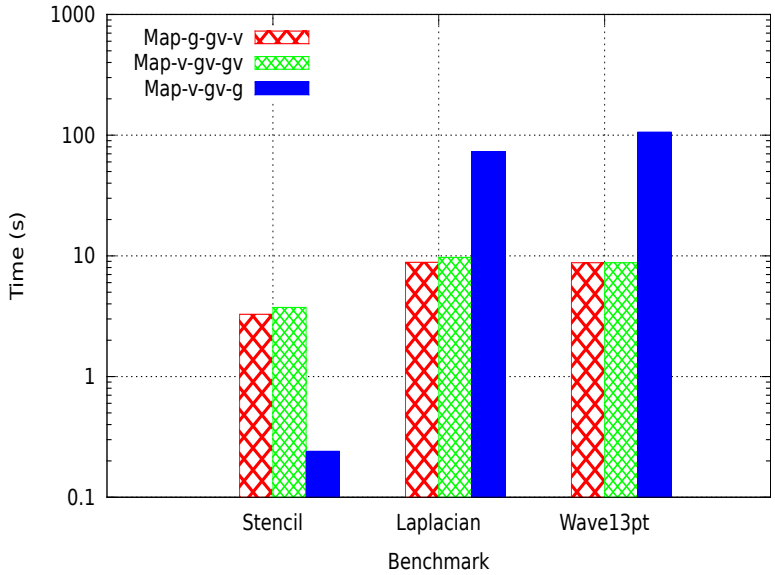


Figure: Performance of Tripe nested loop mapping

Performance Comparison between OpenUH and PGI OpenACC

- PGI 13.6: O0 and O3 optimization
- OpenUH: GCC 4.4.7 (O0) and nvcc 5.0.
- Compare the performance between Map-g-gv and Map-gv-gv loop mappings.
- Kernel time which indicates the efficiency of the kernel code generated by compiler
- Total time which includes the kernel time, data transfer time and the runtime overhead.

Double Nested Loop Mapping Algorithms Performance

PGI VS OpenUH

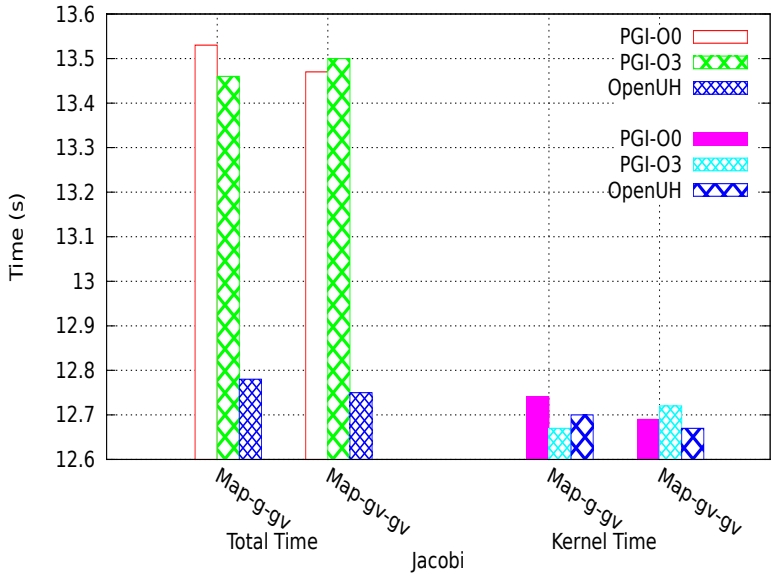


Figure: Jacobi

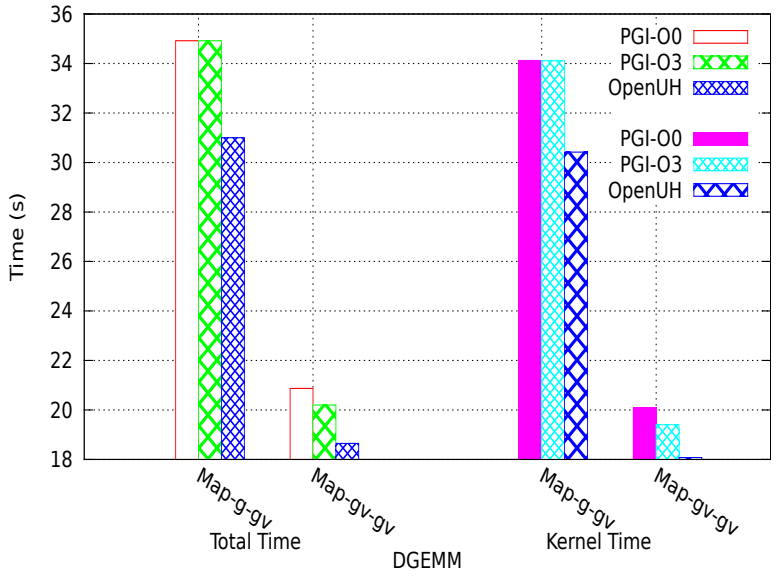


Figure: DGEMM

Double Nested Loop Mapping Algorithms Performance

PGI VS OpenUH

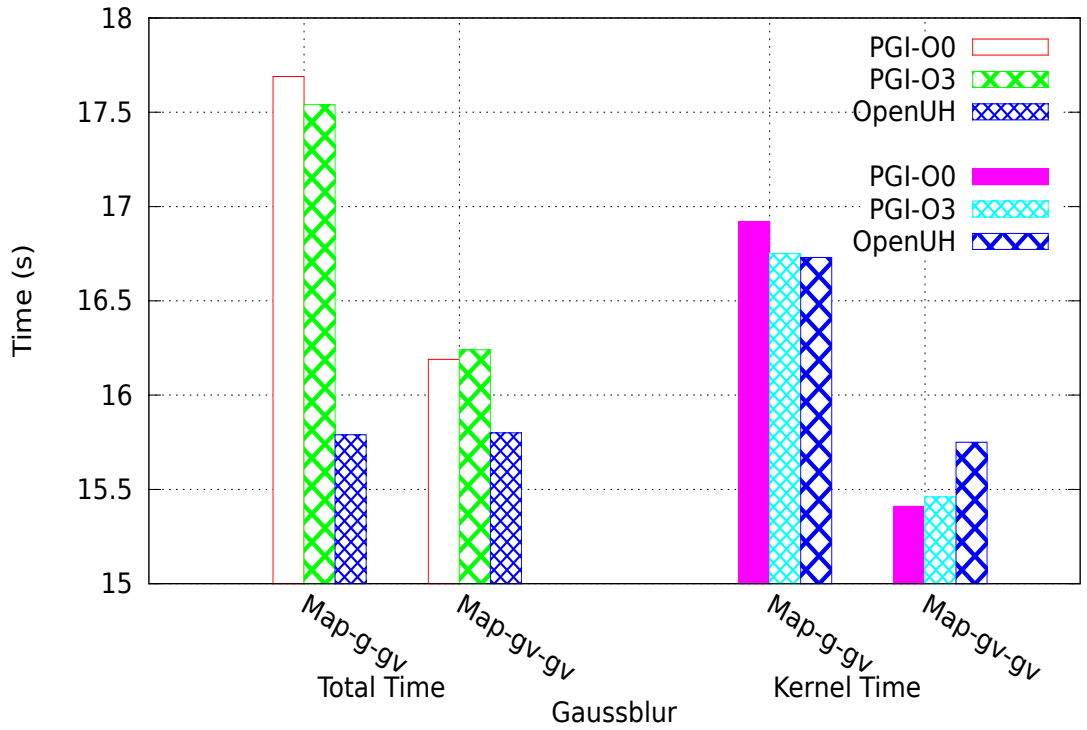


Figure: Gaussblur

Triple Nested Loop Mapping Algorithms Performance

PGI VS OpenUH

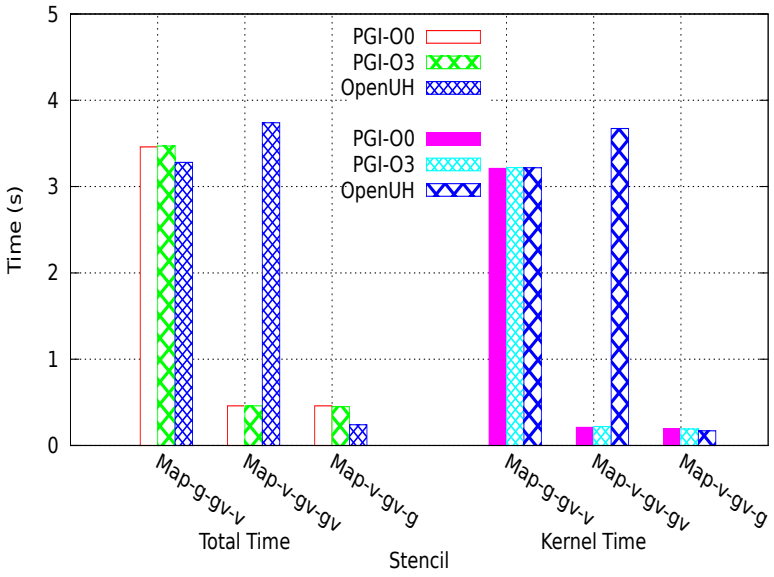


Figure: Stencil

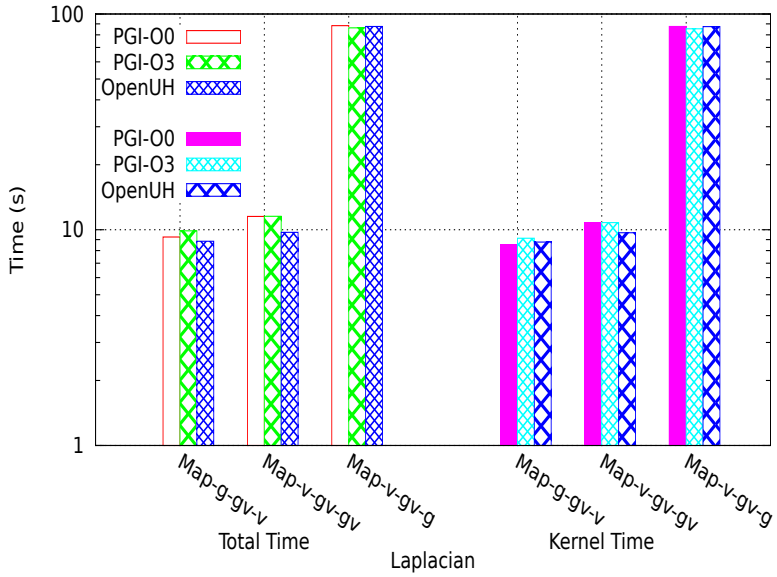


Figure: Laplacian

Triple Nested Loop Mapping Algorithms Performance

PGI VS OpenUH

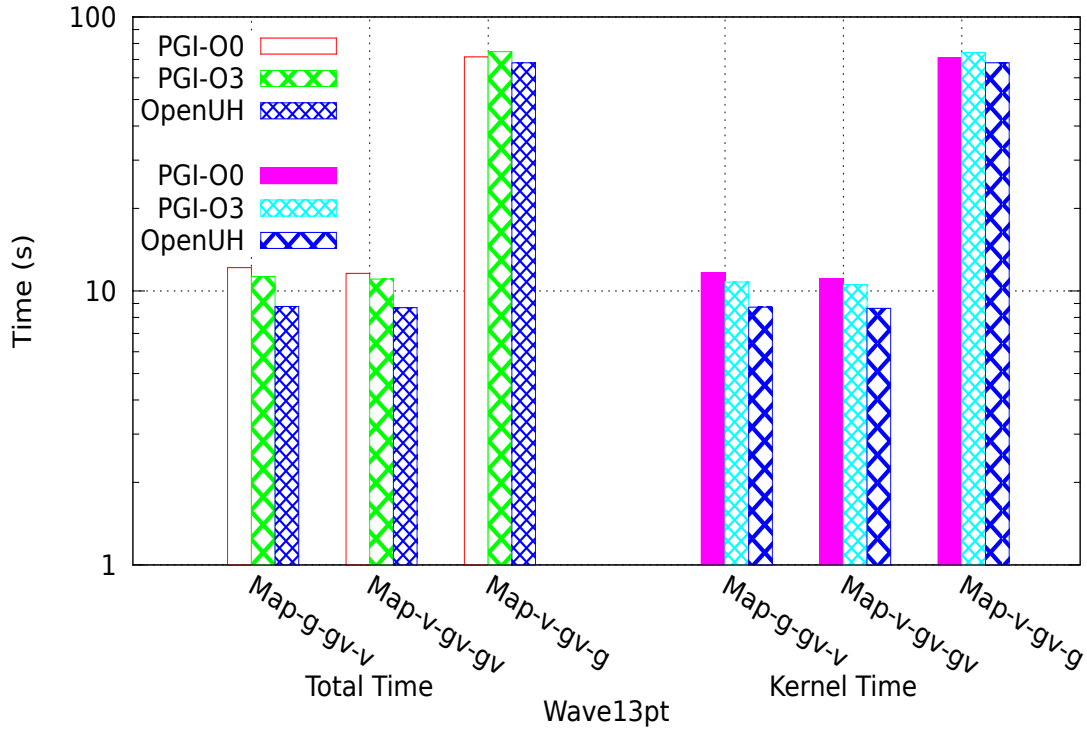


Figure: Wave13pt

- OpenACC compilers
  - Commercial OpenACC compilers from CAPS, PGI, and Cray
  - accULL is another opensource openacc compiler
- Other solutions for GPGPUs
  - KernelGen ports the existing code into Nvidia GPU without the need of adding any directives
  - HiCuda was one of the early directed based programming model for GPGPUs.
  - OpenMPC translates OpenMP code to cuda.

- Conclusion
  - An open-source OpenACC compiler is created using OpenUH compiler framework
  - Loop mapping mechanisms are designed to translate single loop, double and triple nested loop
  - Competitive performance compared to the commercial OpenACC compiler
- Future Work
  - Implement the advanced features such as multi-dimensional array, loop collapse and cache etc.
  - Explore advanced compiler analysis and transformation techniques to further improve the performance