Advanced Computer Architecture
高级计算机体系结构

第8讲：DLP and GPU (3)

GPU在计算流体力学（CFD）中的应用

张曦

DCS5367, 11/23/2021

Self-introduction

- XI ZHANG (张曦)
- Work: Engineer in NSCC-GZ
- Interests: High Performance Computing, Computational Fluid Dynamics
- Contact: xi.zhang@nscc-gz.cn

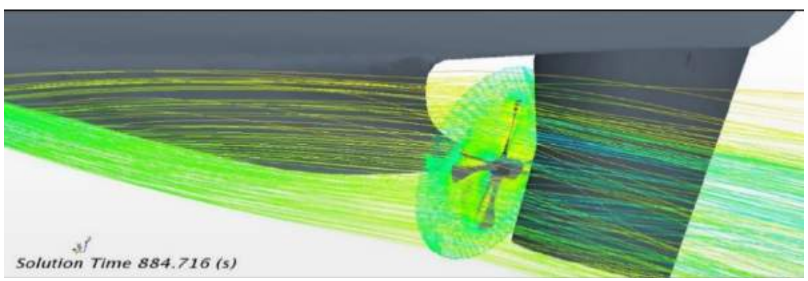
zhangx299@mail.sysu.edu.cn

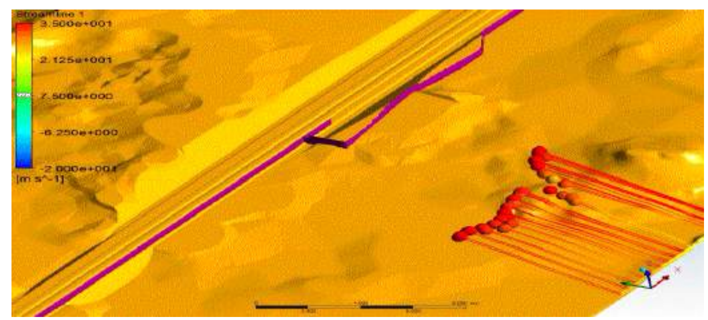
Detail

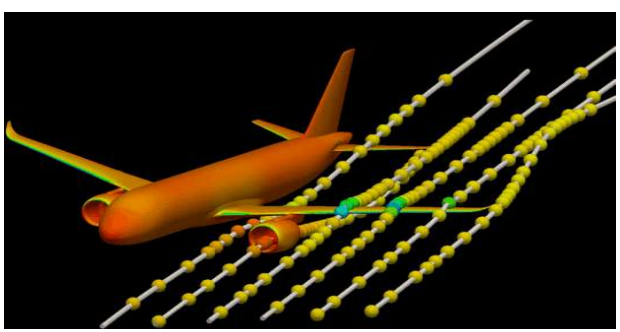
- Background
- Developing Stage
- Optimizations Stage
- Conclusion
- Future work
- Thinking more

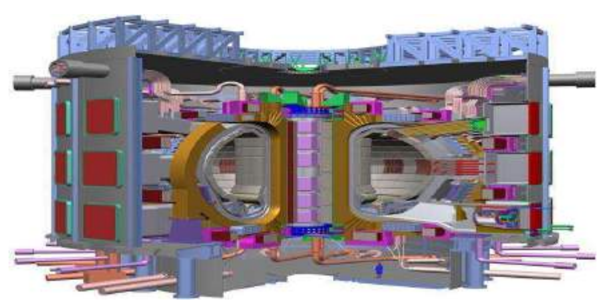
Background

- Scientific computing (科学计算) in Science and Engineering
  - Scientific computing is regarded as the third methodology in science and engineering. (theory 理论 and experiment 实验)
  - Scientific computing is widely used in aerospace science and technology, ocean engineering, nuclear industry, etc.









Background

- Scientific computing in Science and Engineering
  - GPU plays more and more important role in scientific computing.
  - Many High Performance Computing Systems are built with GPU.

天河-1：
4.7 PFLOPs
CPU (飞腾1000)+GPU
(Nvidia Tesla M2050)

Summit：
200 PFLOPs
CPU (Power 9)+GPU
(Nvidia Tesla V100)

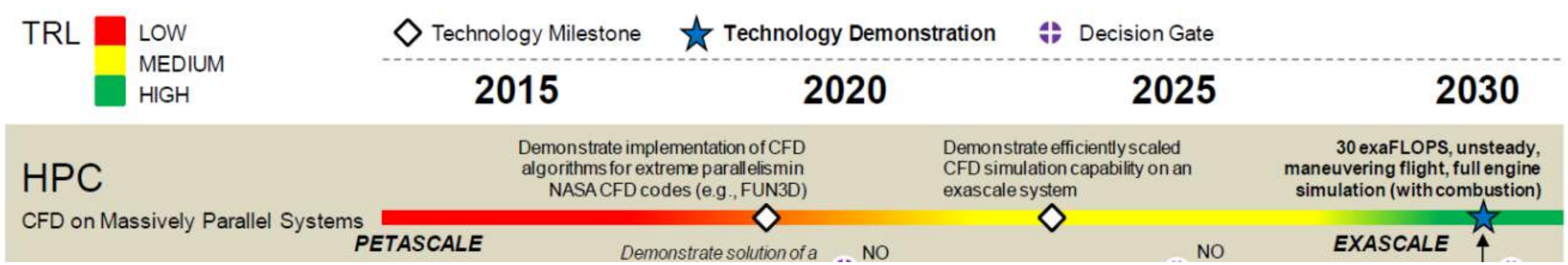
Frontier：
CPU (AMD Zen 3)+GPU
(AMD MI200)

Background

- Computational Fluid Dynamics (CFD)
  - A process of mathematically modeling (数学建模) a physical phenomenon (物理现象) involving fluid flow and solving it numerically (数值求解) using the computational prowess.

$$\frac{\partial u}{\partial t} + \nabla \cdot \mathbf{f}(u, \nabla u) = S(\mathbf{x}, t),$$

  - A cross-discipline subject including mathematics (数学), fluid dynamics (流体力学), and computer science (计算机科学).
  - CFD requires large computing source (E级计算需求)

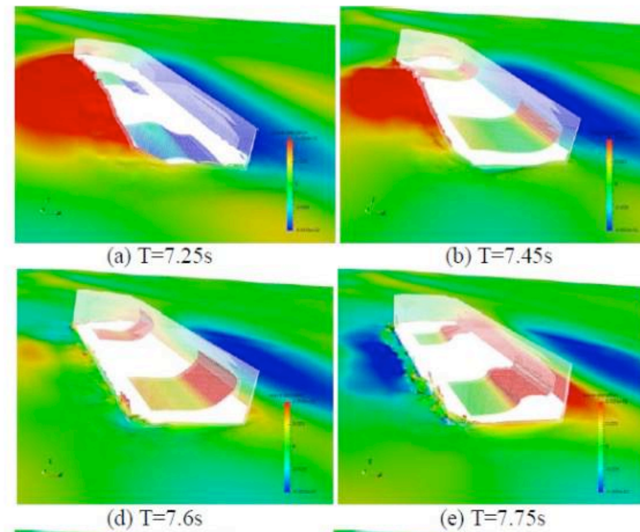


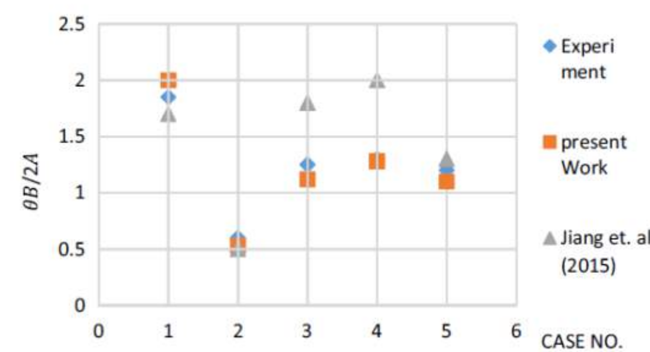
Background

- What is CFD?
  - Quite like Physical Rendering in Computer Graphics

CG

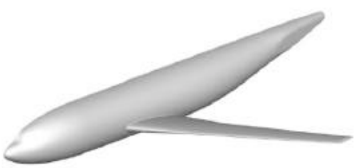
CFD

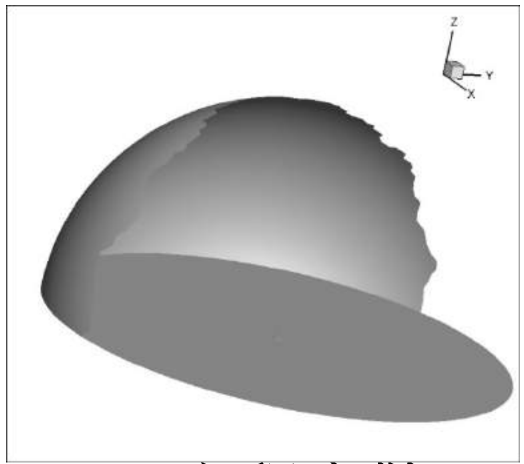


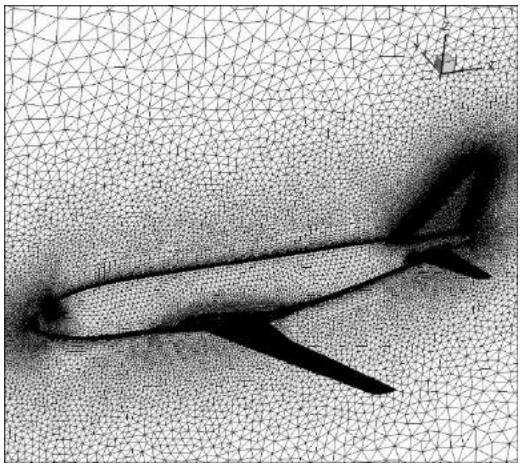


Background

- The process of CFD
  - Domain discretization (计算域离散)







  - Equation discretization (方程离散)

$$\frac{\partial u}{\partial t} + \nabla \cdot \mathbf{f}(u, \nabla u) = S(\mathbf{x}, t),$$

$$\frac{V}{\Delta \tau}\Delta \mathbf{q} + \frac{\partial \hat{\mathbf{R}}}{\partial \mathbf{q}}\Delta \mathbf{q} = -\mathbf{R}(\mathbf{q}^{\mathbf{n}})$$

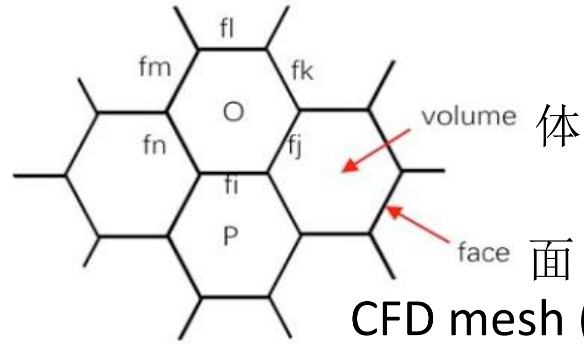
$$\mathbf{q}^{\mathbf{n+1}} = \mathbf{q}^{\mathbf{n}} + \Delta \mathbf{q}$$

Background

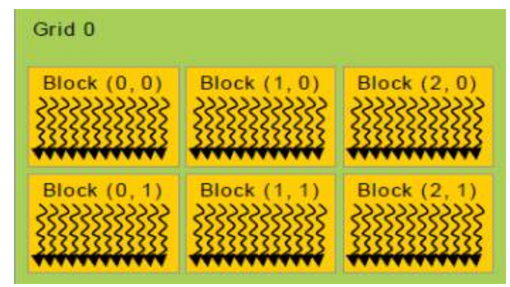
- Difficulties of CFD by GPU
  - Precision (精确): code from CPU to GPU, data dependence (数据依赖)

Algorithm 1 Flux Summation (FS) by FVM face-loop

```
for faceID = 0 to numFaces-1 do
    ownVolID ← owner[faceID]
    ngbVolID ← neighbor[faceID]
    res[ownVolID] ← res[ownVolID]+flux[faceID]
    res[ngbVolID] ← res[ngbVolID]-flux[faceID]
end for
```



CFD mesh (网格)



GPU multi-thread computing

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \dots \\ b_n \end{bmatrix}$$

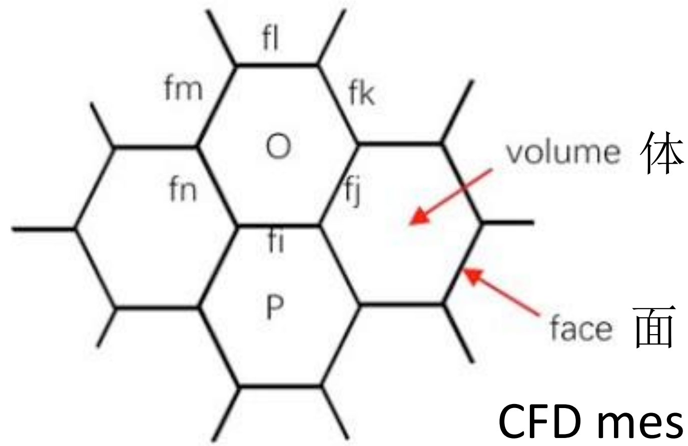
$$A = L + D + U$$

$$(L+D)x^* = b - Ux^n$$

$$(L+U)x^{n+1} = b - Ux^n - Lx^*$$

Background

- Difficulties of CFD by GPU
  - Performance (性能): non-coalescing (非对齐) memory access



CFD mesh (网格)

面编号 face number	0	• • •	1	• • •	f2	• • •	fj	• • •
owner	*owner*[0]	• • •	*owner*[1]	• • •	O	• • •	*owner*[fj	• • •
neighbor	*neighbor*[0]	• • •	*neighbor*[1]	• • •	P	• • •	*neighbor*[fj]	• • •

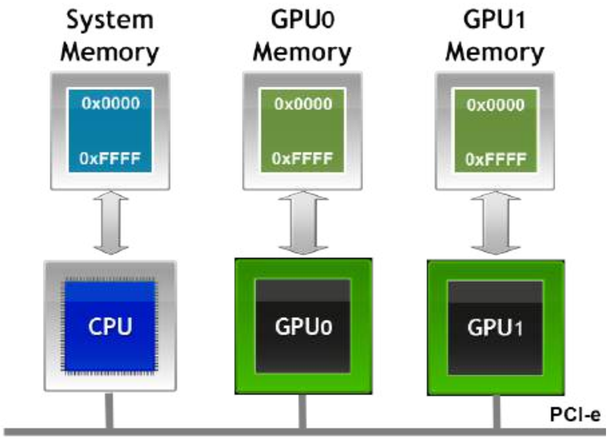
Irregular data storage (不规则存储)

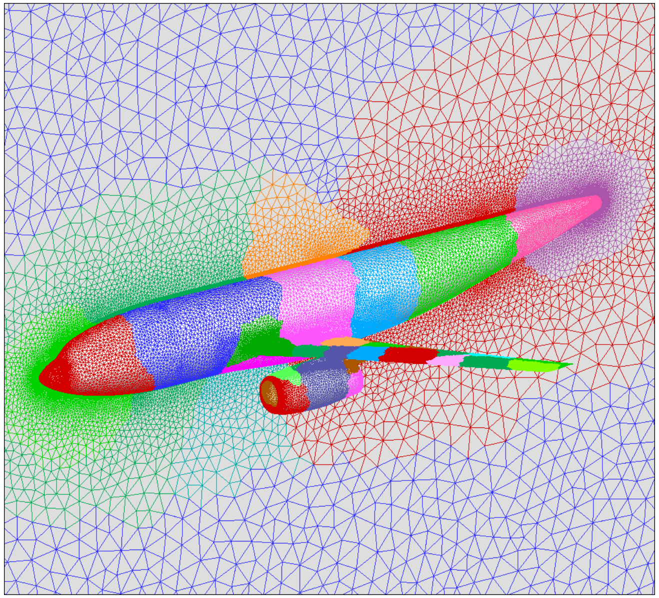
体编号 volume number	0	• • •	1	• • •	O	• • •	P	• • •
residual	*res*[0]	• • •	*res*[1]	• • •	*res*[O]	• • •	*res*[P]	• • •
					res[*owner*[fi]]		*res*[*owner*[fj]]	

Indirect data access (间接访问)

Background

- Difficulties of CFD by GPU
  - Data transfer between Host and GPU
  - Multi-GPU computing(多GPU计算): parallel framework

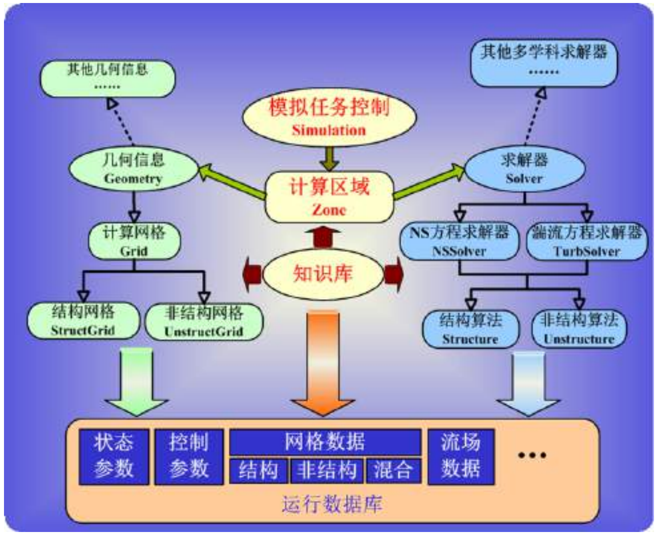




中山大學 SUN YAT-SEN UNIVERSITY

Background

- CFD application: NNW-PHengLEI (风雷)
  - High speed and compressible flow
  - Aeronautics (空气动力学) e.g. plane
  - CPU computing only
  - C++, Object-Oriented
  - Open source (https://forge.osredm.com/projects/p68217053/PHengLEI)
- A similar CFD software Fun3D
  - Spend almost 10 years in R&D on GPU



Developing Stage

- CUDA C Programming
  - Close to loop induced computing (靠近f循环计算部分)
  - Interface functions for calling CUDA kernels (接口函数)
  - Test every CUDA kernel by comparing with CPU results (测试)

```
#ifdef CPURUN
    for ( int m = 0; m < nl; ++ m )
    {
        for ( int iCell = 0; iCell < nTotal; ++ iCell )
        {
            q[m][iCell] = qold[m][iCell];
        }
    }
#endif
#ifdef __CUDACC__
        CallGPULoadQ(nTotal, nl);
        #ifdef CUDAUNITTEST
                TestGPULoadQ(q);
        #endif
#endif
```

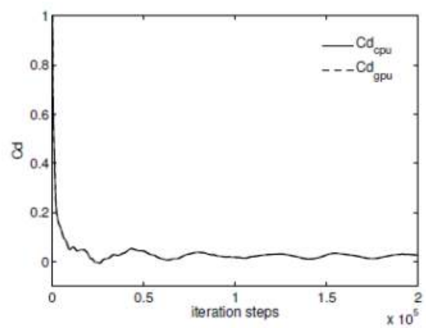
```
void CallGPULoadQ(...) {
    GPULoadQ<<<gridSize, blockSize>>>   (...);
}
__global__ void GPULoadQ(...){
    const int tidx= threadIdx.x;
    const int bidx= blockIdx.x;

    for ( int m = 0; m < nl; ++ m ){
        for ( int icell = bidx* blockDim.x+ tidx;
                    icell < nTotal;
                        icell += gridDim.x* blockDim.x){
            q[m*nTotal+ icell] = qold[m*nTotal+icell];
        }
    }
}
```

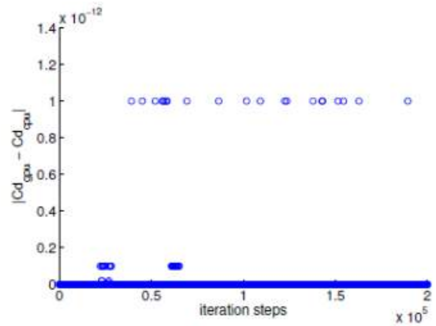
截图(Alt + A)

Developing Stage

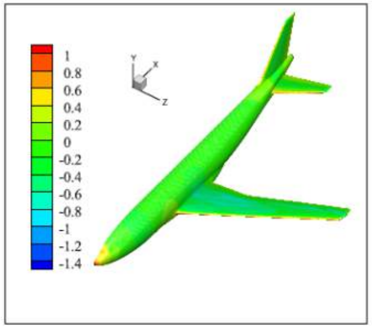
- What can induce error on GPU computing?
  - Loop order on GPU (乱序执行循环)
  - CUDA supported optimizations such as MAD (乘加操作)
  - Some CUDA supported mathematical functions such as pow (幂次运算)



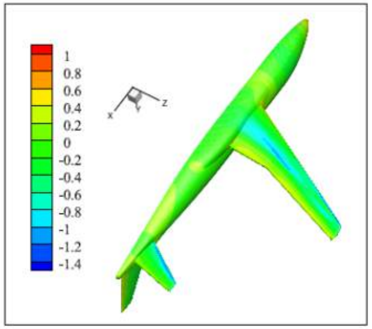
(a) Cd



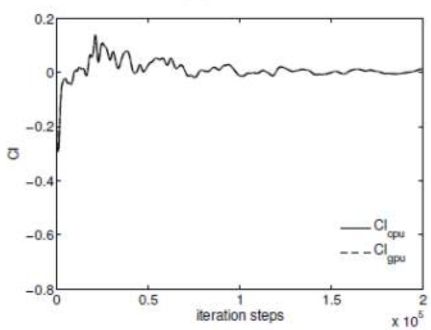
(c) $|Cd_{CPU} - Cd_{GPU}|$



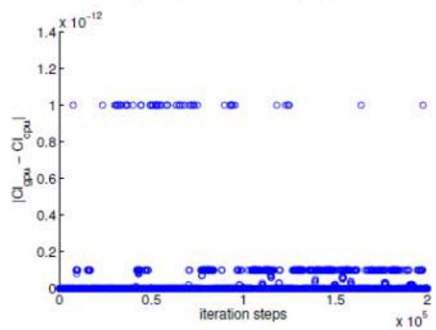
(a) Cp on top side of Chnt (CPU)



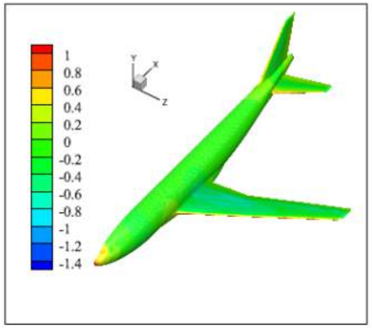
(c) Cp on bottom side of Chnt (CPU)



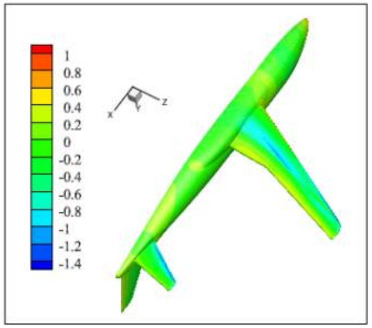
(b) Cl



(d) $|Cl_{CPU} - Cl_{GPU}|$



(b) Cp on top side of Chnt (GPU)

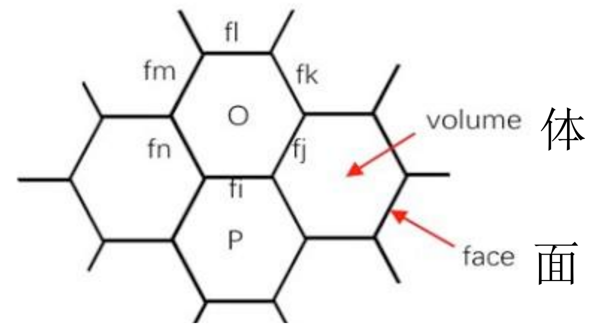


(d) Cp on bottom side of Chnt (GPU)

Optimization Stage

- Atomic operations or graph coloring for resolving data dependence
  - Atomic operations: hardware supported method
  - Graph coloring: software supported method

```
for faceID = nBoundFace to nTotalFace-1 do
    Le ← leftCellOfFace[faceID]
    Re ← rightCellOfFace[faceID]
    for eqnID = 0 to numEqn - 1 do
        setAdd(res[eqnID][Le], flux[eqnID][faceID])
        setAdd(res[eqnID][Re], flux[eqnID][faceID])
    end for
end for
```



Algorithm 4 Summation of Flux by atomic operation (SF-AT)

```
<GPU kernel Begin>
threadID←threadIdx.x+blockIdx.x*blockDim.x
for faceID = nBoundFace+threadID to nTotalFace-1 do
    Le ← leftCellOfFace[faceID]
    Re ← rightCellOfFace[faceID]
    for eqnID = 0 to numEqn - 1 do
        atomicAdd(res[eqnID*nTotalCell+Le],
            flux[eqnID*nTotalFace+faceID])
        atomicAdd(res[eqnID*nTotalCell+Re],
            flux[eqnID*nTotalFace+faceID])
    end for
    setAdd(faceID, blockDim.x*gridDim.x)
end for
<GPU kernel End>
```

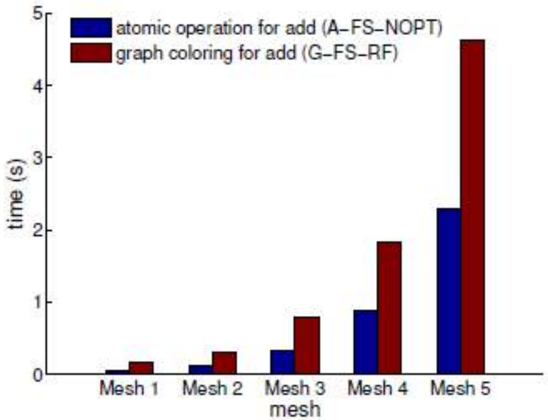
Algorithm 3 Summation of Flux by graph coloring(SF-GC)

```
Reorder flux according to faceGroup
for groupID = 0 to nGroups-1 do
    numFacesInGroup←numFaceOfGroup[groupID]
    groupStart←offsetFaceGroup[groupID]
    <GPU kernel Begin>
    threadID←threadIdx.x+blockIdx.x*blockDim.x
    for faceGroupID = threadID to numFacesInGroup-1 do
        groupFaceID←groupStart+faceGroupID
        faceID←faceGroup[groupFaceID]
        Le←leftCellOfFace[faceID]
        Re←rightCellOfFace[faceID]
        for eqnID = 0 to numEqn - 1 do
            setAdd(res[eqnID*nTotalCell+Le],
                flux[eqnID*nTotalFace+faceID])
            setAdd(res[eqnID*nTotalCell+Re],
                flux[eqnID*nTotalFace+faceID])
        end for
        setAdd(faceGroupID, blockDim.x*gridDim.x)
    end for
    <GPU kernel End>
end for
```

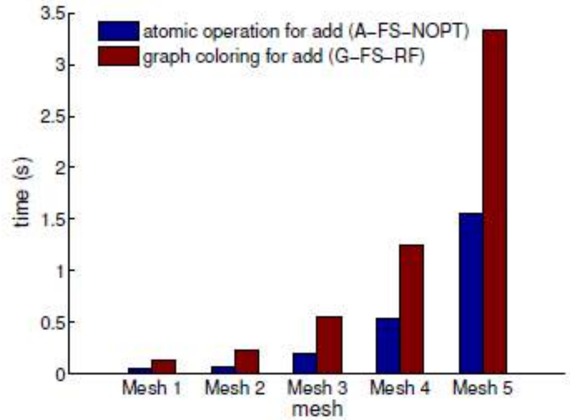
SUN YAT-SEN UNIVERSITY 中山大學

Optimization Stage

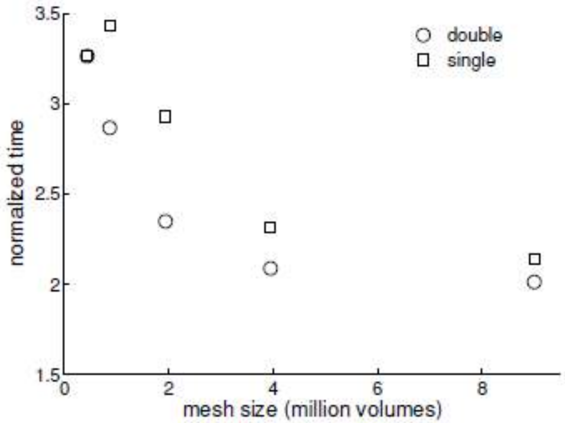
- Atomic operations V.S. graph coloring



(a) double



(b) single



(c) time ratio

Optimization Stage

- Data dependence resolving in LU-SGS scheme (数据依赖性)
  - Data dependence in LU-SGS
  - Multi-color LU-SGS

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \dots \\ b_n \end{bmatrix}$$

$$A = L + D + U$$

L-SGS $\quad (L+D)x^* = b - Ux^n$

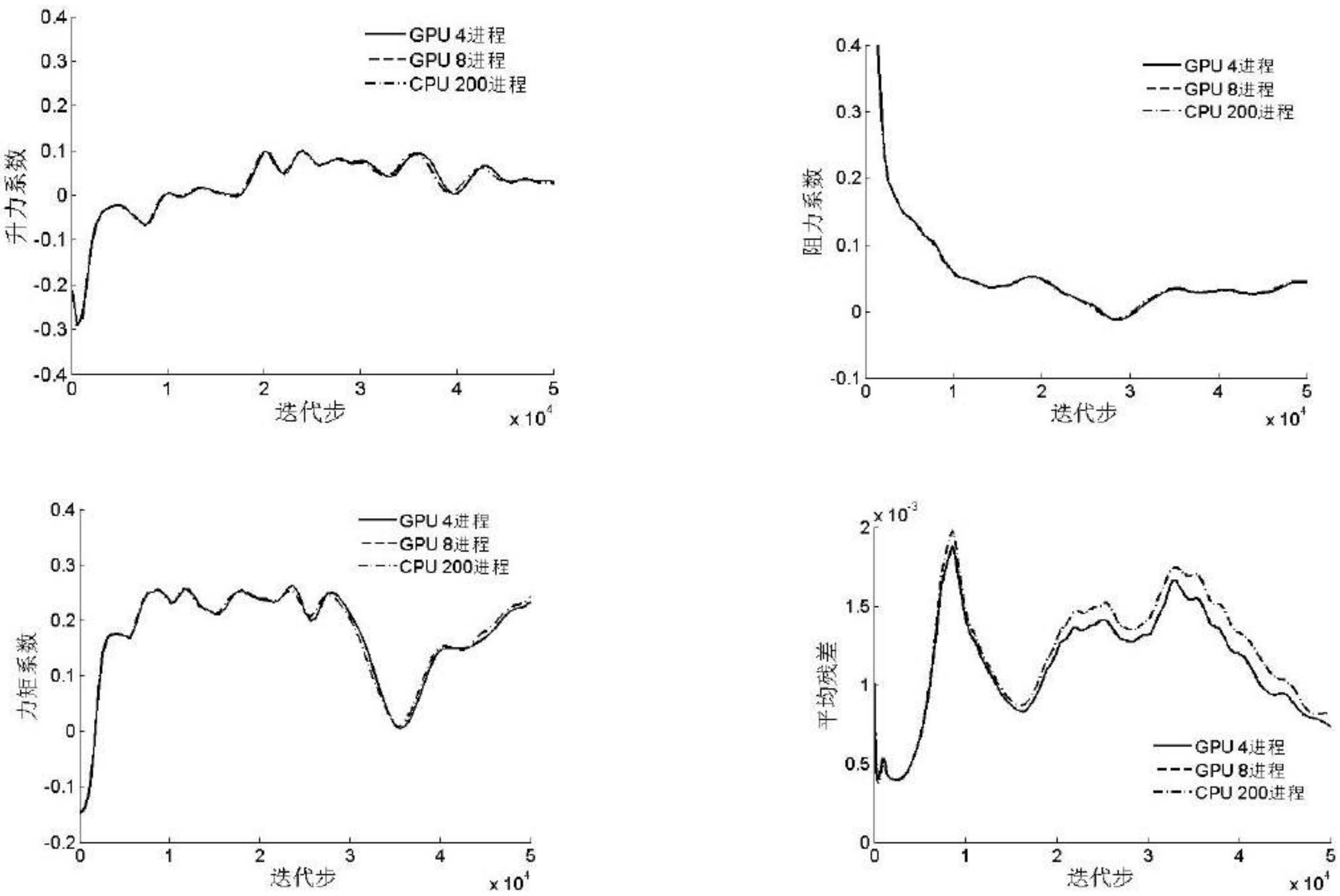
U-SGS $\quad (L+U)x^{n+1} = b - Ux^n - Lx^*$

```
for colorID = 0 to numColors-1 do
  numColorGroup ← colorGroupNum[colorID]
  posiColorGroup ← colorGroupPosi[colorID]
  <GPU kernel Begin>
  for offset = 0 to numColorGroup do
    cellID ← colorGroup[posiColorGroup+offset]
    LowerSweepOnOneCell(cellID)
  end for
  <GPU kernel End>
end for
```

```
for colorID = numColors-1 to 0 do
  numColorGroup ← colorGroupNum[colorID]
  posiColorGroup ← colorGroupPosi[colorID]
  <GPU kernel Begin>
  for offset = 0 to numColorGroup do
    cellID ← colorGroup[posiColorGroup+offset]
    UpperSweepOnOneCell(cellID)
  end for
  <GPU kernel End>
end for
```

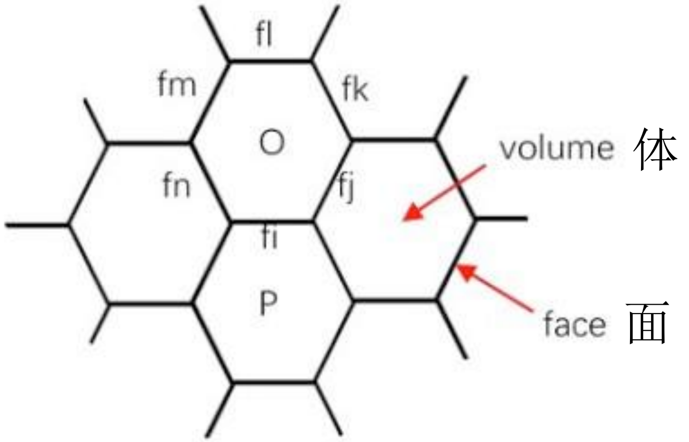
Optimization Stage

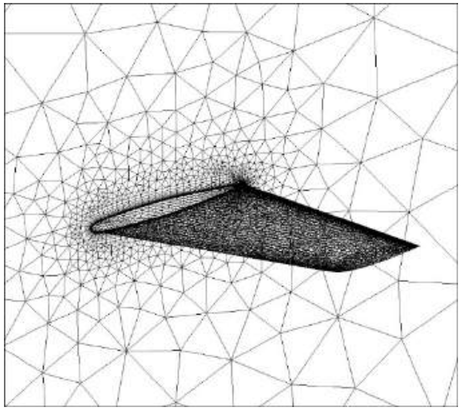
- Result of Multi-dolor LU-SGS

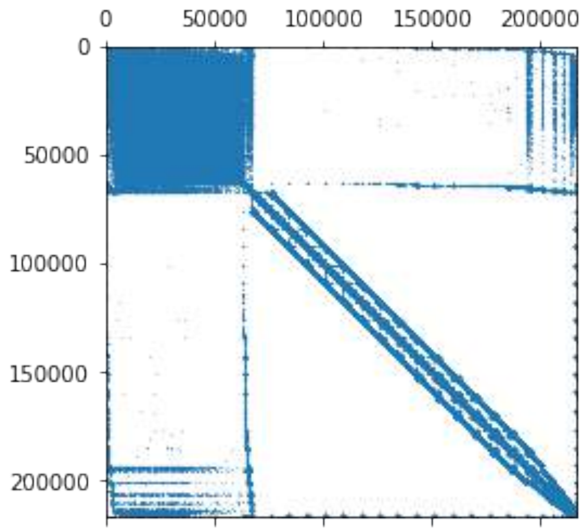


Optimization Stage

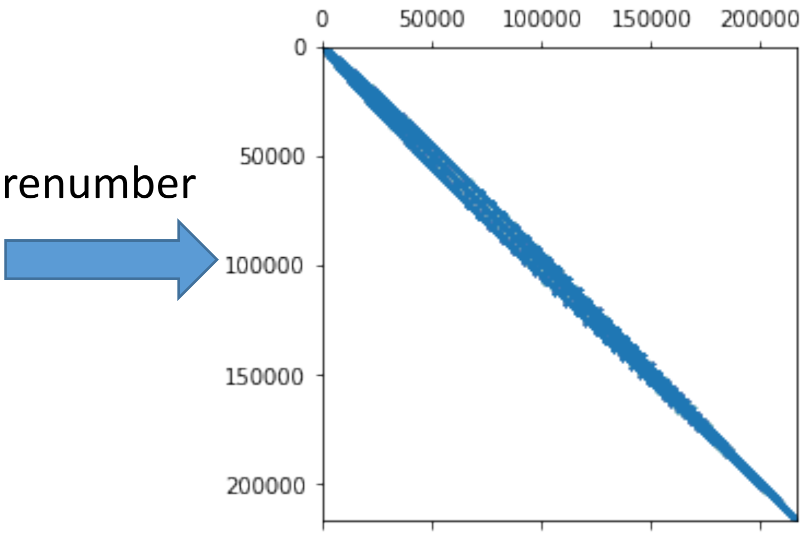
- Volume renumber (体编号重排序)
  - Reverse Cuthill-Mckee (RCM)
  - Reduce adjacent matrix bandwidth
  - Easing non-coalescing data access







renumber

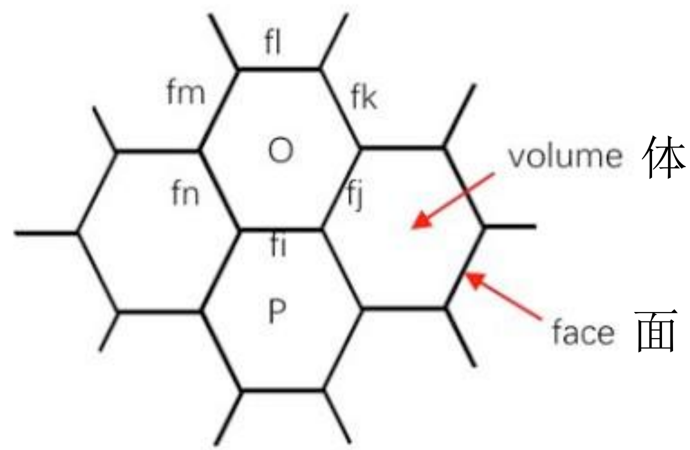


Optimization Stage

- Face renumber (面编号重排序)
  - Renumber face in a volume (以体为单位顺序重排面编号)
  - Optimizing data locality (提高数据局部性)

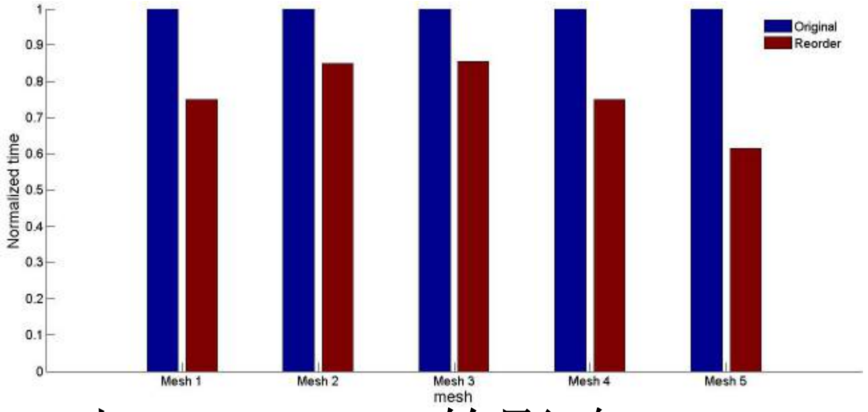
算法 1 面编号排序算法

```
Reorder faces in cellFace by ascending order
mapFace ← -1
labelFace ← nBoundFace-1
for cellID = 0 to nTotalCell-1 do
    offset ← offsetCellFace[cellID]
    numFaces ← numFaceOfCell[cellID]
    for faceInCell = 0 to numFaces-1 do
        faceID ← cellFace[offset+faceInCell]
        if faceID > nBoundFace-1 then
            if mapFace[faceID] == -1 then
                labelFace ← labelFace+1
                mapFace[faceID] ← labelFace
            end if
        end if
    end for
end for
Update mesh connectivity information by mapFace
```

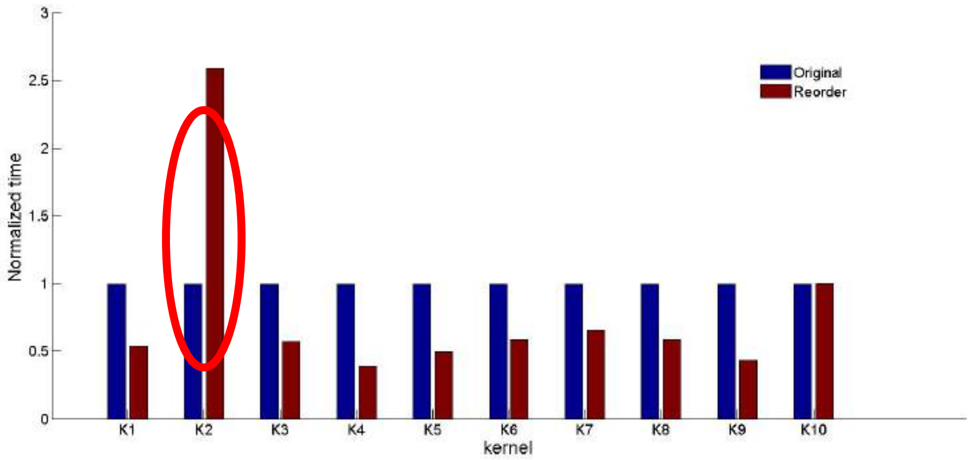


Optimization Stage

- Results of volume and face renumber
  - Overall Performance (对GPU程序整体性能的影响)



  - GPU kernels (对GPU kernels的影响)

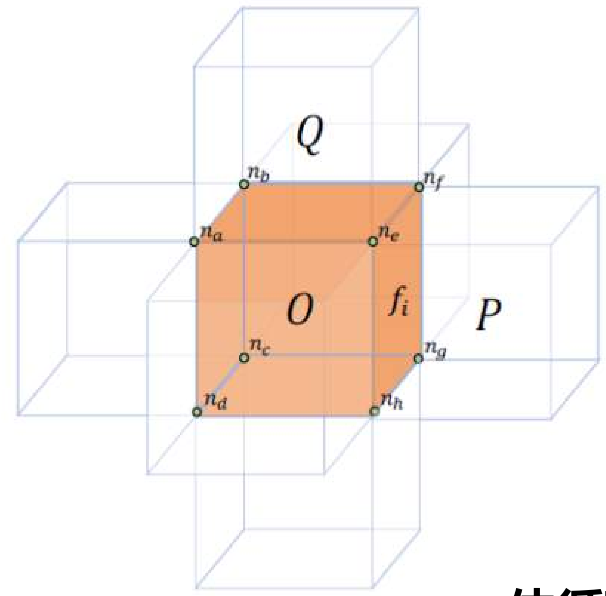


Optimization Stage

- Loop mode adjust 1 (循环模式调整)
  - Data interpolation
  - Can a face loop be replaced?

```
<GPU kernel Begin>
threadID←threadIdx.x+blockIdx.x*blockDim.x
for faceID = nBoundFace+threadID to nTotalFace-1 do
    Le ← leftCellOfFace[faceID]
    Re ← rightCellOfFace[faceID]
    faceNodeStart ← offsetFaceNode[faceID]
    numNodeInFace←numNodeOfFace[faceID]
    for faceNodeID = 0 to numNodeInFace-1 do
        nodeID ← faceNodes[faceNodeStart+faceNodeID]
        for eqnID = 0 to numEqn - 1 do
            atomicAdd(qNode[eqnID*nTotalNode+nodeID],
            q[eqnID*nTotalCell+Le])
        end for
        atomicAdd(tNode[nodeID], t[Le])
        atomicAdd(nCount[nodeID], 1)
        for eqnID = 0 to numEqn - 1 do
            atomicAdd(qNode[eqnID*nTotalNode+nodeID],
            q[eqnID*nTotalCell+Re])
        end for
        atomicAdd(tNode[nodeID], t[Re])
        atomicAdd(nCount[nodeID], 1)
    end for
    setAdd(faceID, blockDim.x*gridDim.x)
end for
<GPU kernel End>
```

面循环

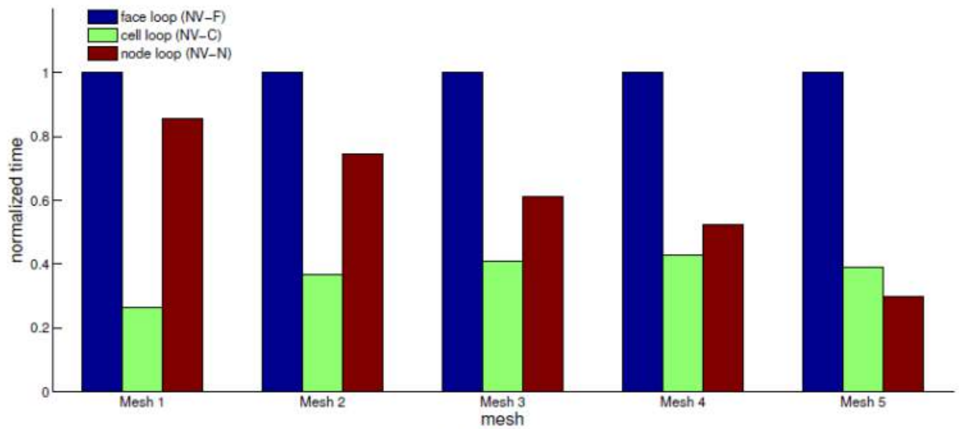


体循环

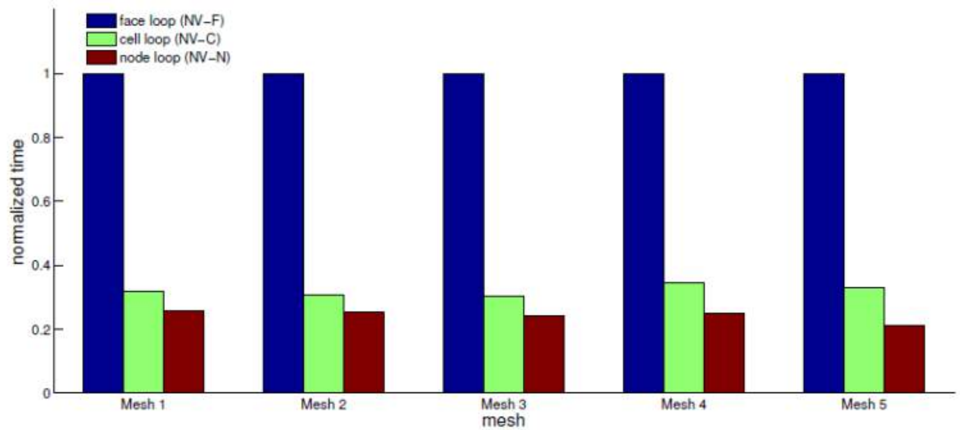
```
<GPU kernel Begin>
threadID←threadIdx.x+blockIdx.x*blockDim.x
for cellID = threadID to nTotalCell-1 do
    cellNodePosi ← offsetCellNode[cellID]
    for offset = 0 to numNodeOfCell[faceID] - 1 do
        nodeID ← cellNodes[cellNodePosi+offset]
        accessFrequency←
        cellNodeCount[cellNodePosi+offset]
        atomicAdd(qNode[eqnID*nTotalNode+nodeID],
        accessFrequency*q[eqnID*nTotalCell+cellID])
    end for
    setAdd(cellID, blockDim.x*gridDim.x)
end for
<GPU kernel End>
```

Optimization Stage

- Results of loop mode adjust



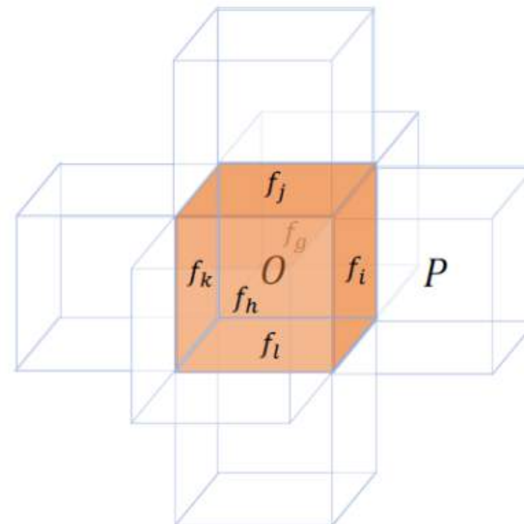
(a) V100



(b) K80

Optimization Stage

- Loop mode adjust 2 (循环模式调整)
  - Data comparison
  - Can a face loop be replaced?



Algorithm 10 Local pressure comparing by face loop (LPC-F)

```
<GPU kernel Begin>
threadID←threadIdx.x+blockIdx.x*blockDim.x
for faceID = threadID+nBoundFace to nTotalFace-1 do
    Le ← leftCellOfFace[faceID]
    Re ← rightCellOfFace[faceID]
    atomicMin(pMin[Le], pressure[Re])
    atomicMax(pMax[Le], pressure[Re])
    atomicMin(pMin[Re], pressure[Le])
    atomicMax(pMax[Re], pressure[Le])
    setAdd(faceID, blockDim.x*gridDim.x)
end for
<GPU kernel End>
```

面循环

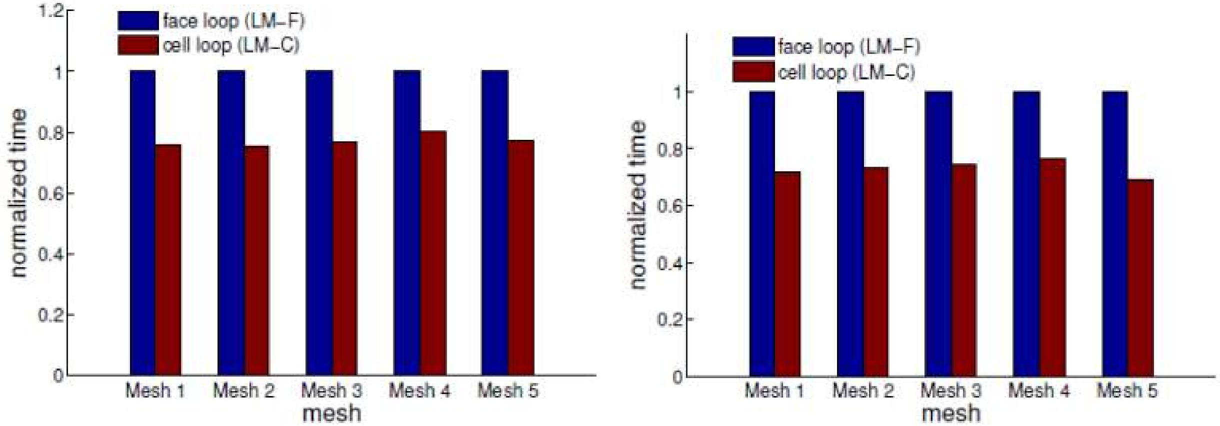
Algorithm 11 Local pressure comparing by cell loop (LPC-C)

```
<GPU kernel Begin>
threadID←threadIdx.x+blockIdx.x*blockDim.x
for cellID = threadID to nTotalCell-1 do
    numCell←numCellCell[cellID]
    cellStart←offsetCellCell[cellID]
    for cellInCellID = 0 to numCell-1 do
        cellCellID←cellCell[cellStart+cellInCellID]
        setCompMin(pMin[cellID], pressure[cellCellID])
        setCompMax(pMax[cellID], pressure[cellCellID])
    end for
    setAdd(cellID, blockDim.x*gridDim.x)
end for
<GPU kernel End>
```

体循环

Optimization Stage

- Results of loop mode adjust



(a) V100 (b) K80

Optimization Stage

- Nested loop split (嵌套循环拆分)
  - Loop on geometry
  - Loop on dimensions

```
for  faceID = nBoundFace to nTotalFace-1 do
    Le ← leftCellOfFace[faceID]
    Re ← rightCellOfFace[faceID]
    for  eqnID = 0 to numEqn - 1 do
        setAdd(res[eqnID][Le], flux[eqnID][faceID])
        setAdd(res[eqnID][Re], flux[eqnID][faceID])
    end for
end for
```

面循环

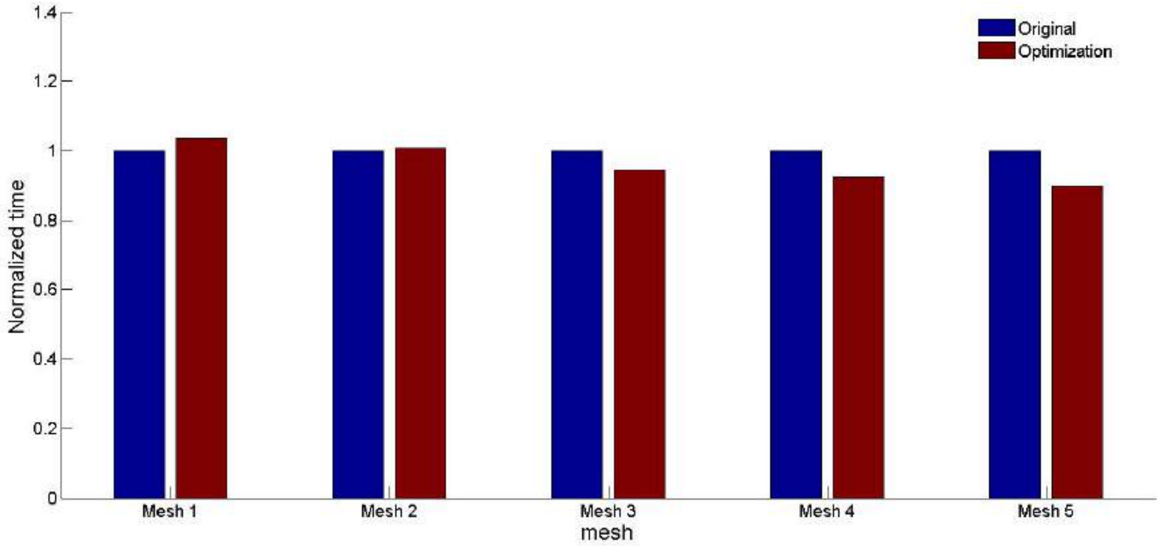
方程数量循环

```
<GPU kernel Begin>
threadID←threadIdx.x+blockIdx.x*blockDim.x
for  faceID = nBoundFace+threadID to nTotalFace-1 do
    Le ← leftCellOfFace[faceID]
    Re ← rightCellOfFace[faceID]
    for  eqnID = 0 to numEqn - 1 do
        atomicAdd(res[eqnID*nTotalCell+Le],
        flux[eqnID*nTotalFace+faceID])
        atomicAdd(res[eqnID*nTotalCell+Re],
        flux[eqnID*nTotalFace+faceID])
    end for
    setAdd(faceID, blockDim.x*gridDim.x)
end for
<GPU kernel End>
```

```
for  eqnID = 0 to numEqn - 1 do
    <GPU kernel Begin>
    threadID←threadIdx.x+blockIdx.x*blockDim.x
    for faceID=nBoundFace+threadID to nTotalFace-1 do
        Le ← leftCellOfFace[faceID]
        Re ← rightCellOfFace[faceID]
        atomicAdd(res[eqnID*nTotalCell+Le],
        flux[eqnID*nTotalFace+faceID])
        atomicAdd(res[eqnID*nTotalCell+Re],
        flux[eqnID*nTotalFace+faceID])
        setAdd(faceID, blockDim.x*gridDim.x)
    end for
    <GPU kernel End>
end for
```

Optimization Stage

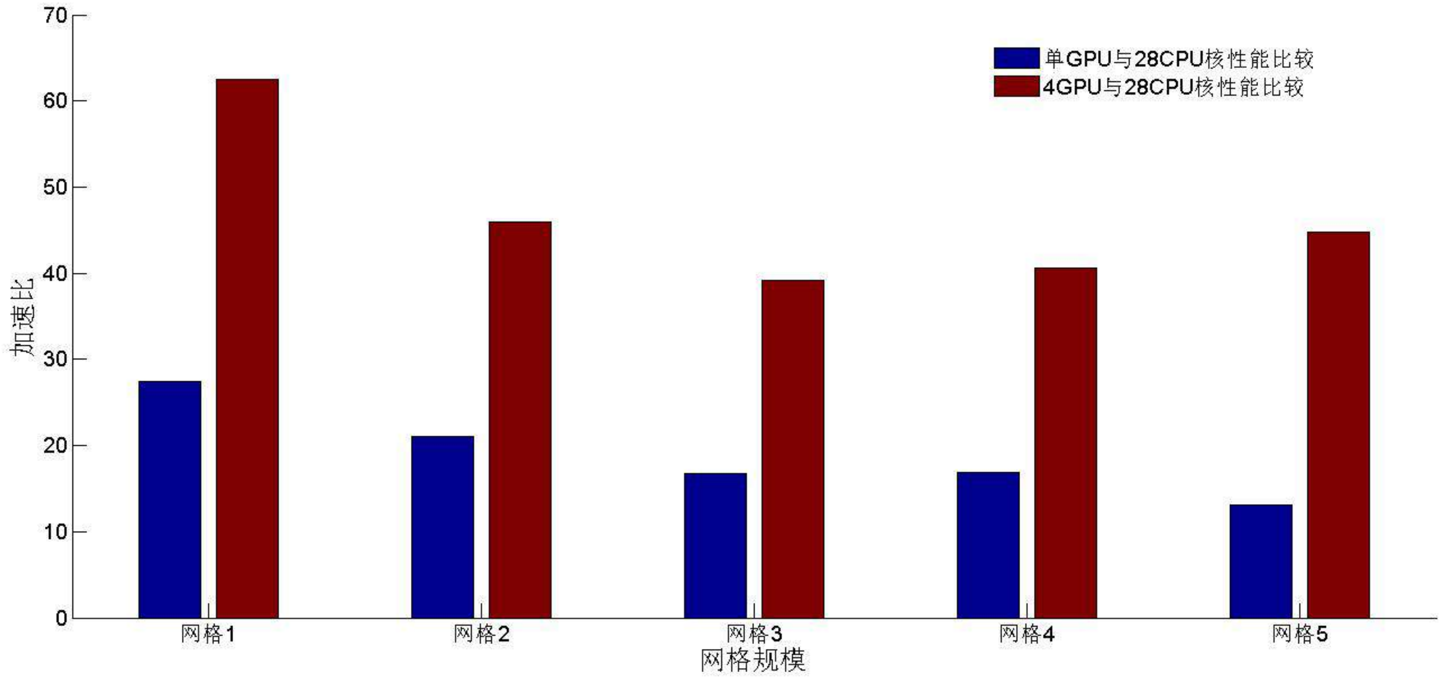
- Results of Nested loop split



SUN YAT-SEN UNIVERSITY

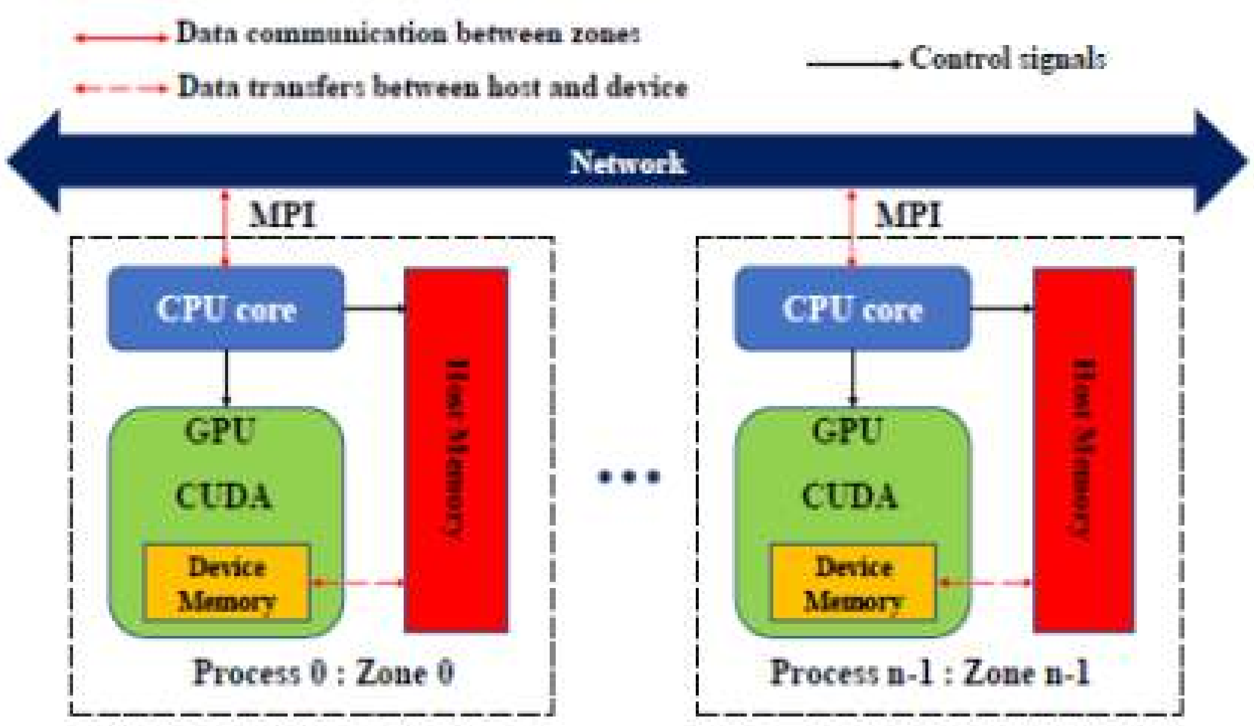
Optimization Stage

- Performance comparison of single GPU and 28 CPU cores
  - GPU: 4 x Nvidia Tesla V100
  - CPU: 2 x Intel Xeon Gold 6132



Optimization Stage

- Multi-GPU computing (多GPU计算)
  - MPI-CUDA parallel framework
  - CPU is only used for controlling GPU



Optimization Stage

- Pack and Unpack MPI data on GPU

Algorithm 13 Pack MPI data on GPU

```
Get dataSend and dataIF by data name
for ngbZoneID = 0 to numNgbZone do
    startFace←startFaceForSend[ngbZoneID]
    startSend←startDataSend[ngbZoneID]
    numNgbFace←nIFaceOfNgbZone[ngbZoneID]
    <GPU kernel Begin>
    threadID←threadIdx.x+blockIdx.x*blockDim.x
    for faceID = threadID to numNgbFace do
        sendID←faceForSend[startFace+faceID]
        for eqnID= 0 to numEqn do
            ngbZoneFaceID←startSend+
            eqnID*numNgbFace+faceID
            interfaceID←eqnID*nInterfaceTotal+sendID
            dataSend[ngbZoneFaceID]←dataIF[interfaceID]
        end for
        setAdd(faceID, blockDim.x*gridDim.x)
    end for
    <GPU kernel End>
end for
```

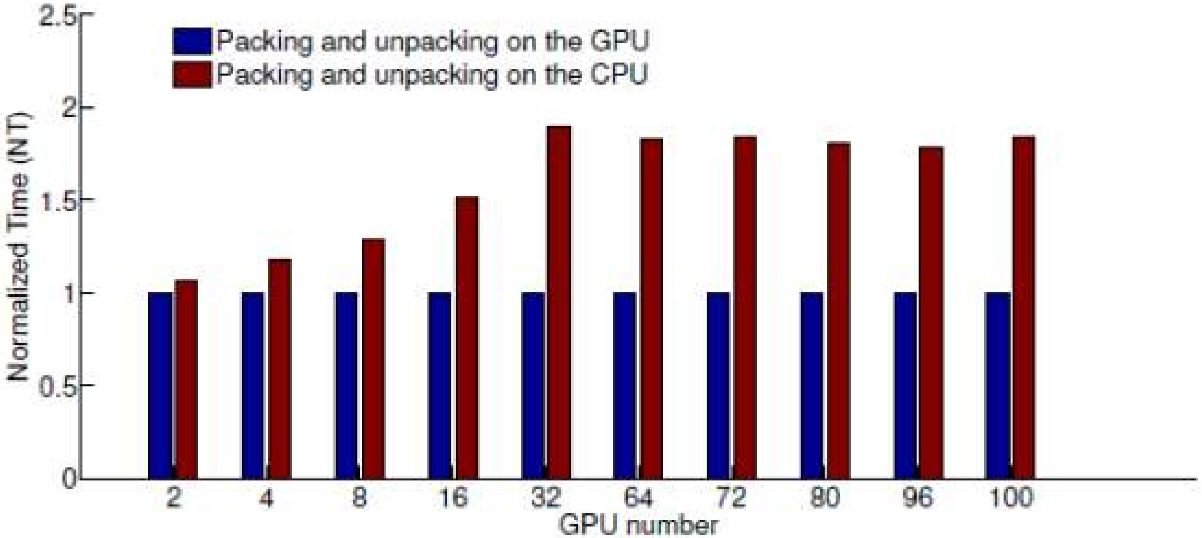
Algorithm 14 Unpack MPI data on GPU

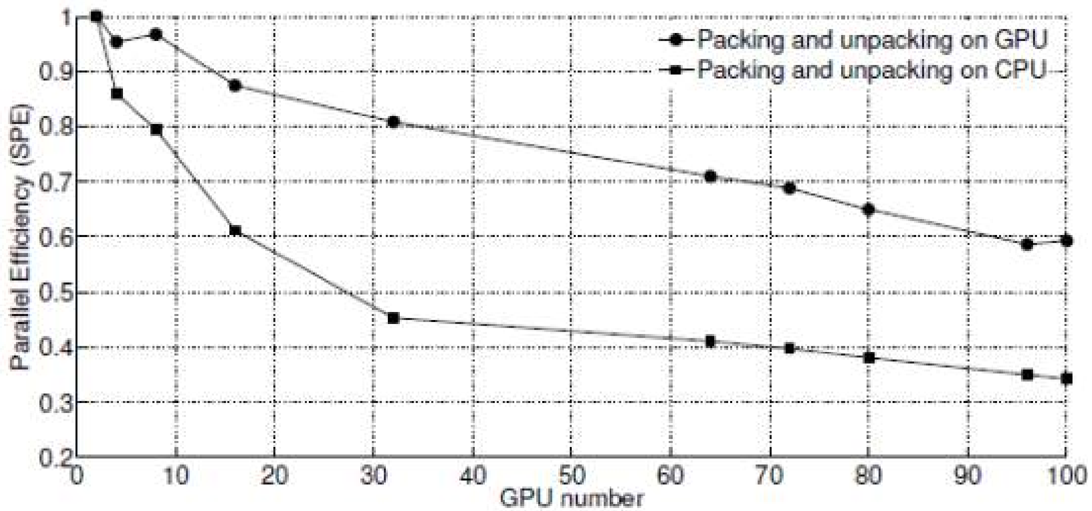
```
Get dataSend and dataIF by data name
for ngbZoneID = 0 to numNgbZone do
    startFace←startFaceForRecv[ngbZoneID]
    startRecv←startDataRecv[ngbZoneID]
    numNgbFace←nIFaceOfNgbZone[ngbZoneID]
    <GPU kernel Begin>
    threadID←threadIdx.x+blockIdx.x*blockDim.x
    for faceID = threadID to numNgbFace do
        recvID←faceForRecv[startFace+faceID]
        for eqnID= 0 to numEqn do
            ngbZoneFaceID←startRecv+
            eqnID*nIFaceOfNgbZone+faceID
            interfaceID←eqnID*nInterfaceTotal+recvID
            dataIF[interfaceID]←dataRecv[ngbZoneFaceID]
        end for
        setAdd(faceID, blockDim.x*gridDim.x)
    end for
    <GPU kernel End>
end for
```

中山大學 SUN YAT-SEN UNIVERSITY

Optimization Stage

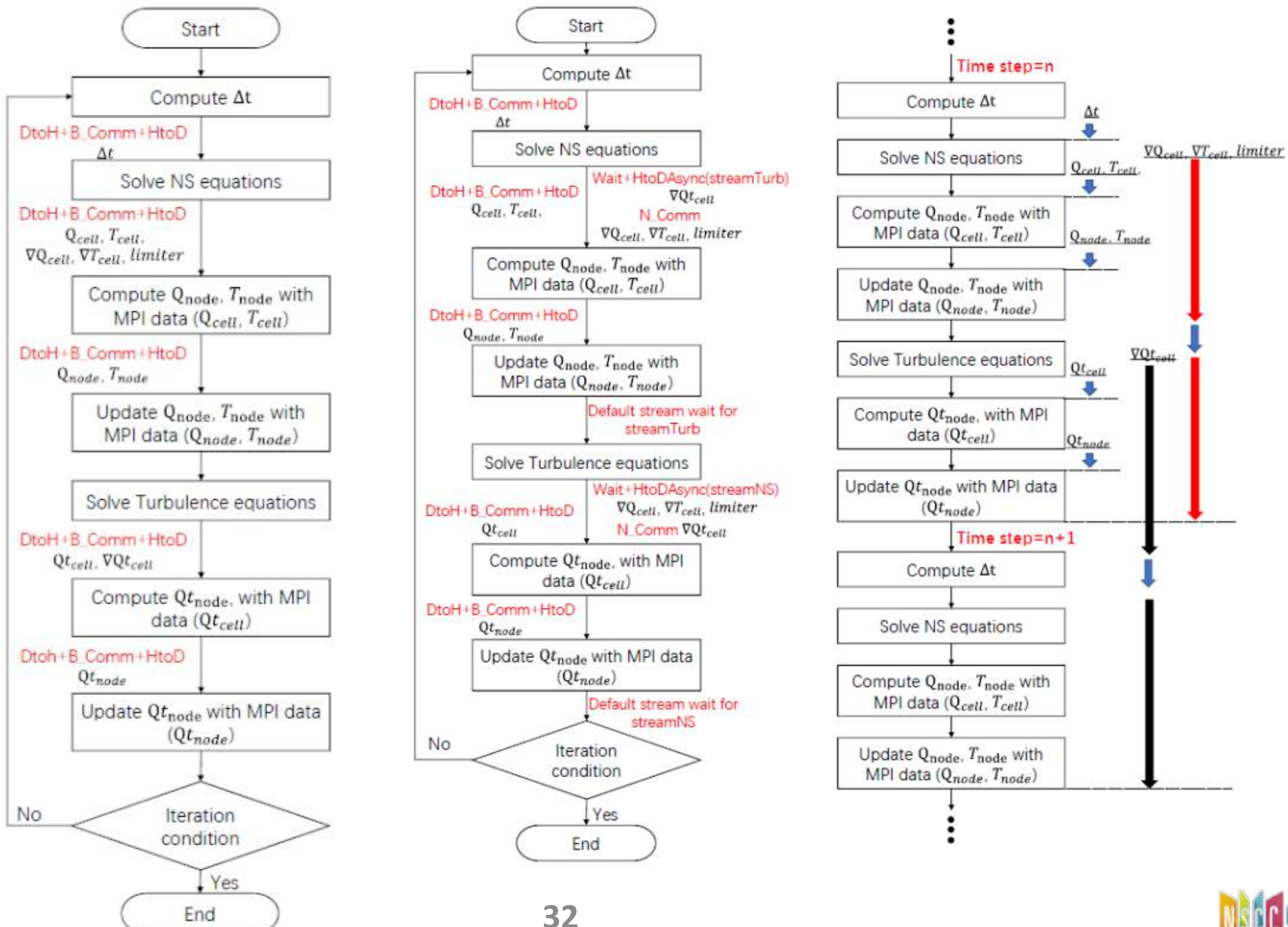
- Result of pack and Unpack MPI data on GPU





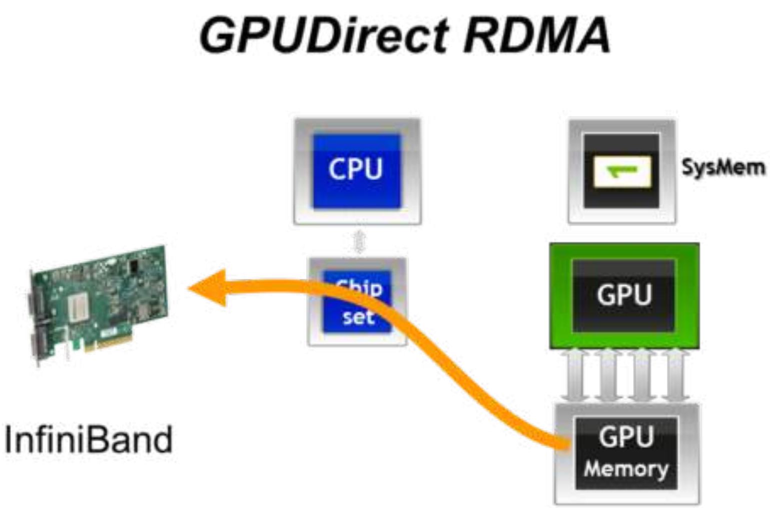
Optimization Stage

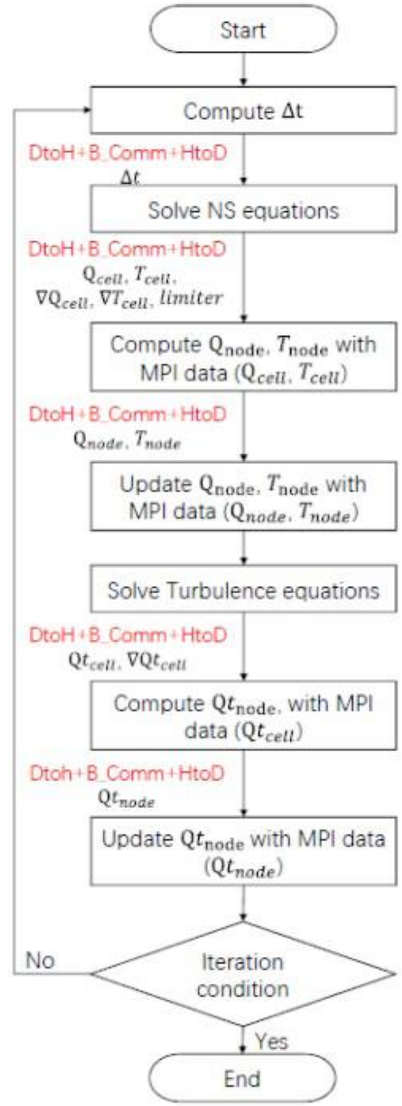
- Communication and computing overlap

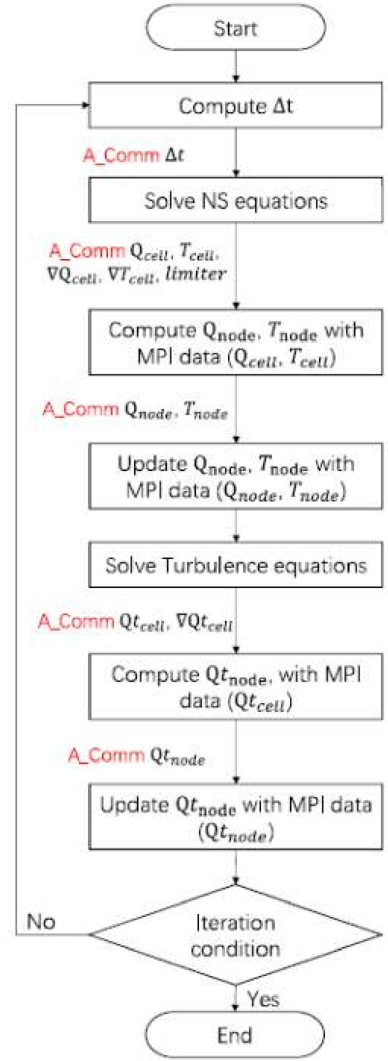


Optimization Stage

- CUDA-AWARE-MPI

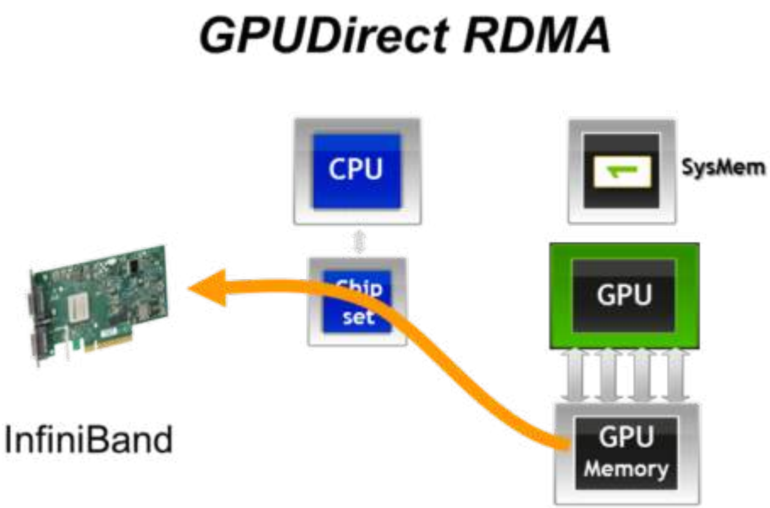


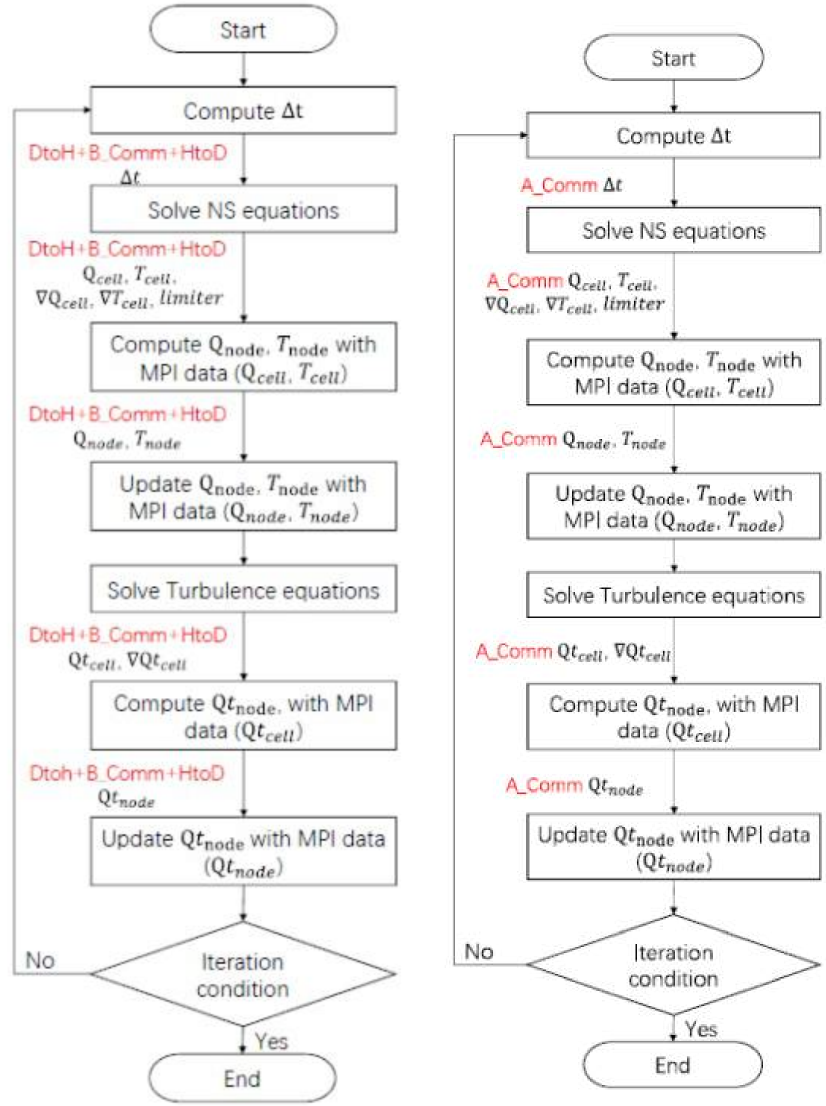




Optimization Stage

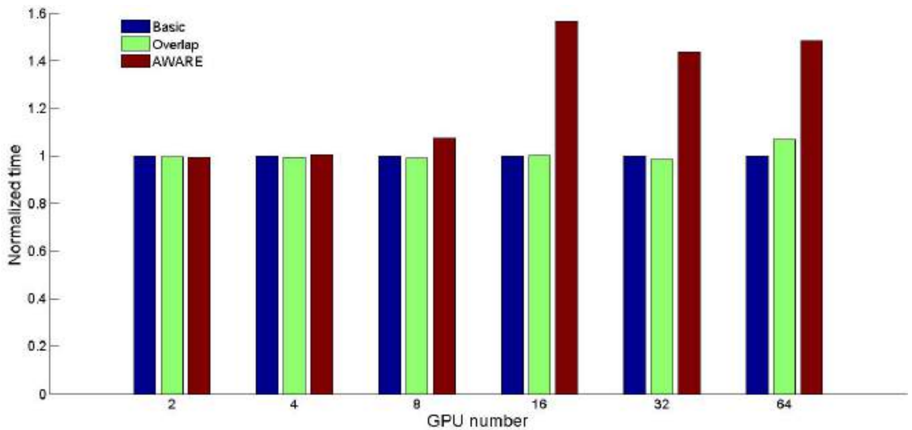
- CUDA-AWARE-MPI

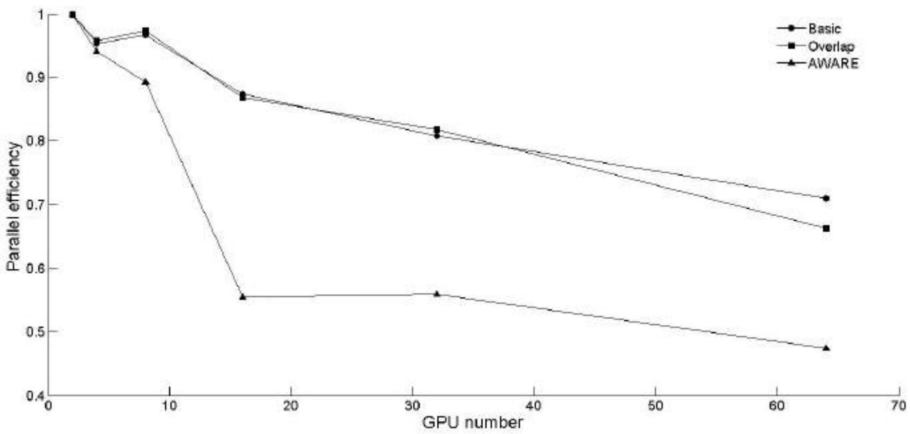




Optimization Stage

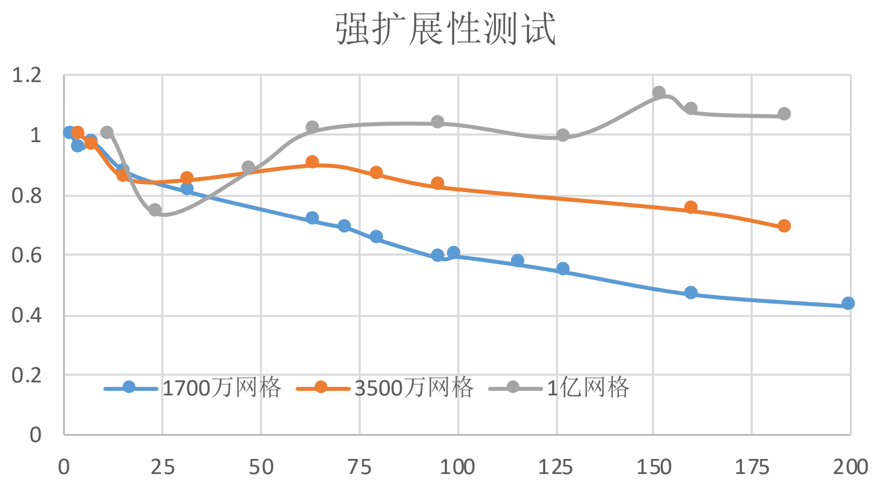
- Result of 3 MPI-CUDA parallel framework

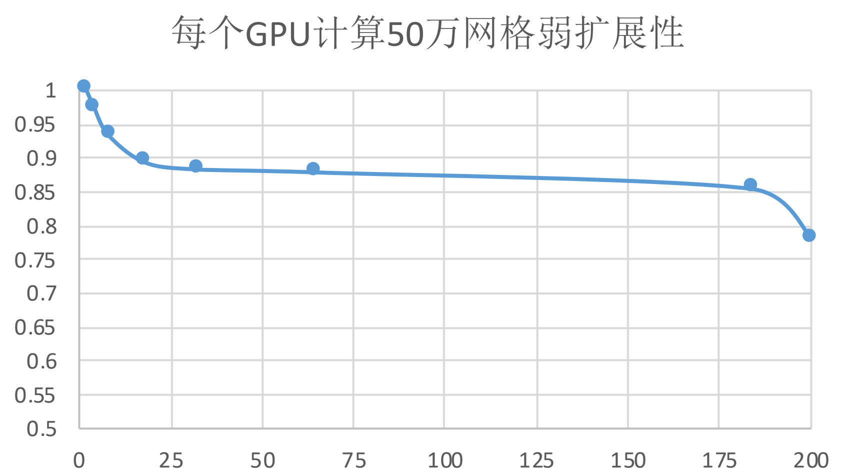




Optimization Stage

- Strong and weak scaling test (扩展性测试)





Conclusion

- 高保真异构程序架构确保计算精度
- Multi-color LU-SGS解决数据依赖性
- 提高数据局部性优化：网格编号重排、循环模式优化、循环嵌套拆分、原子操作解决资源竞争
- 多GPU计算：开发MPI-CUDA并行框架及其改进版本；实现数据在GPU上的打包、解包
- 实现了GPU的高精度计算；最多实现了200块GPU的并行计算，并行效率较高

Future Work

- Multi-color LU-SGS的进一步优化
- Mixed Precision computing
- Tensor core
- CFD+AI

Thinking more …

- 硬件
  - 领域订制硬件 FPGA, RISC-V等
- 软件
  - 可移植性
  - 可维护性
  - 领域订制软件

Thank you for your listening

联系方式：zhangx299@mail.sysu.edu.cn
xi.zhang@nscc-gz.cn