GPU acceleration of the FINE/FR CFD solver in a heterogeneous environment with OpenACC directives

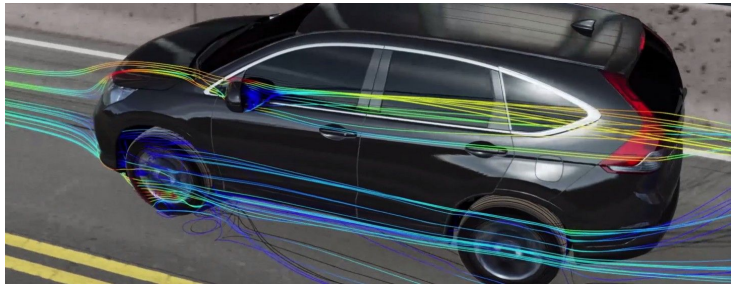
Xiaomeng 'Shine' Zhai[1], David Gutzwiller[1], Kunal Puri[2], Charles Hirsch[2]

1:Numeca-USA 2:Numeca-International

GPU acceleration of FINE/FR CFD solver with OpenACC

Background: CFD & FINE/FR

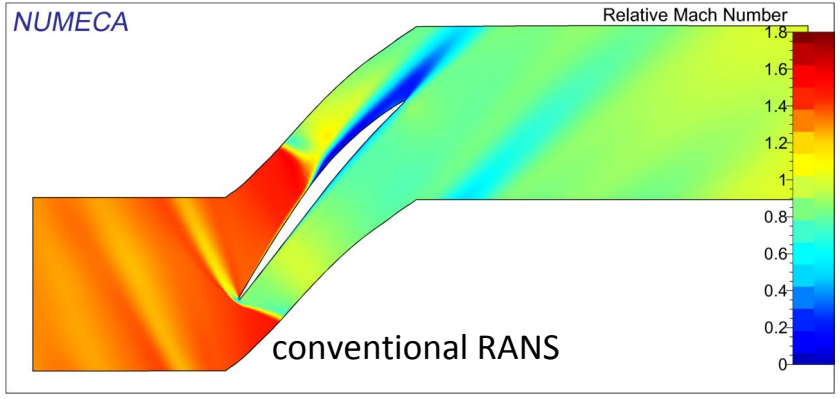
- Computational Fluid Dynamics
  - numerically solve governing equations of fluid motion, and quantitatively predict fluid-flow phenomena

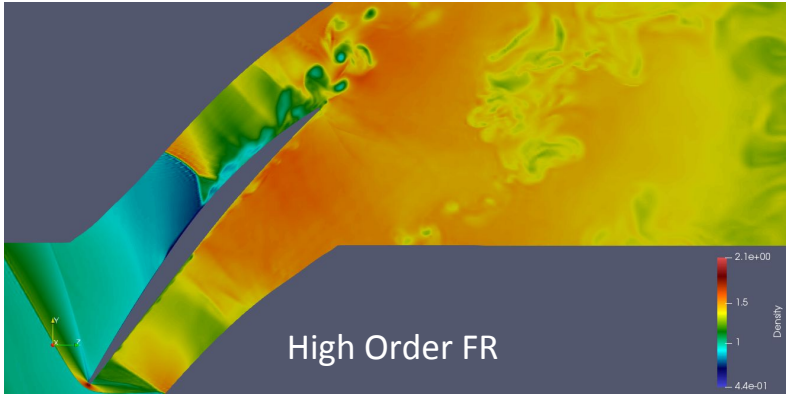


streamlines around a Honda CR-V

- FINE/FR solver

  Based on High Order Flux-Reconstruction method [Huynh 2007]
  - high accuracy compared to conventional methods, capturing fine-scale motions
  - dense math calculations, compact computational stencil; well suited for GPU





GPU acceleration of FINE/FR CFD solver with OpenACC

Current Research: FINE/FR & INCITE

- OpenACC: a natural choice for Numeca
  - 2X+ GPU speedup in time-to-solution for Fortran based FINE/TURBO [2015 WACCPD]
  - Cost-effective to adapt existing/legacy applications for GPU; performance portable
  - Prototyping with C++ based FINE/FR shows good potential for GPU acceleration

- To continue research into high fidelity industrial simulations, NUMECA was selected for a US DOE INCITE project
  - “Towards Understanding Instability Mechanisms of Axial Compressors”
  - 305,000 node hours on OLCF SUMMIT, 2020 - 2021
  - Targeting the high resolution simulation of rotating stall in an axial compressor
  - Each node: 2 IBM POWER 9 22-core CPUs + 6 Nvidia V100 GPUs
    - faster simulation turnaround depends on efficient use of GPUs

GPU acceleration of FINE/FR CFD solver with OpenACC

Current Research: FINE/FR & INCITE

This presentation will focus on the rapid implementation of high performance CPU+GPU support with OpenACC and cuBLAS in preparation for leadership scale computations on SUMMIT.

GPU acceleration of FINE/FR CFD solver with OpenACC

FINE/FR: Programming Model

- Distributed Parallel MPI framework
  - Static domain decomposition via parMETIS
  - Distributed parallel checkpointing and restart
  - Solver iteration loop (>95% of the execution time)
    - Multiple calls to Intel MKL BLAS matrix multiplication routines
    - Dozens of additional correction and calculation loops
  - Demonstrated strong scalability on tens of thousands of cores

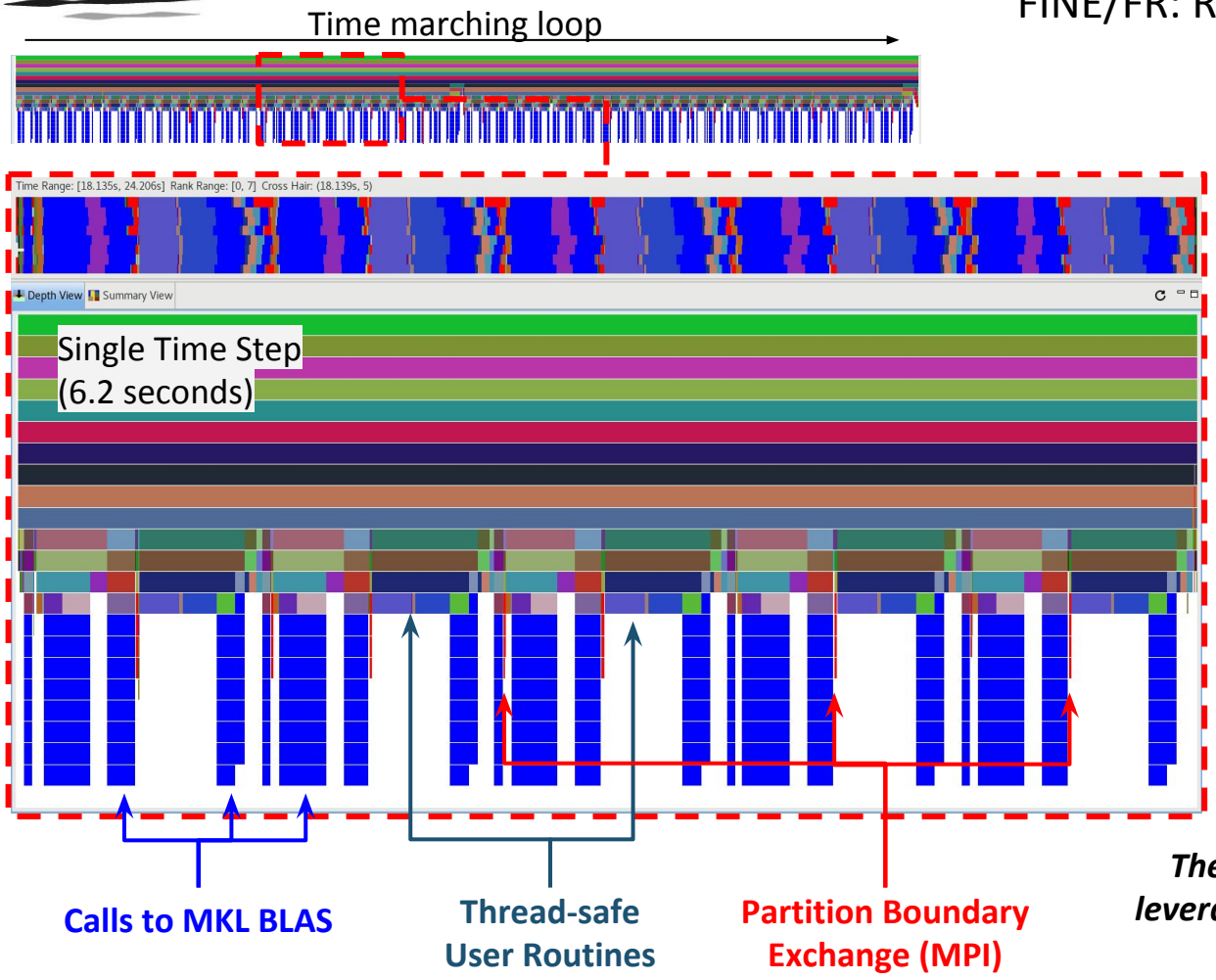
Natural Target for GPU Acceleration

- Language: C++11
  - Object oriented throughout; extensive templatization in the core solver algorithms
  - Limited polymorphism, mostly outside of the iteration loop
  - With some efforts, OpenACC is capable of handling this code

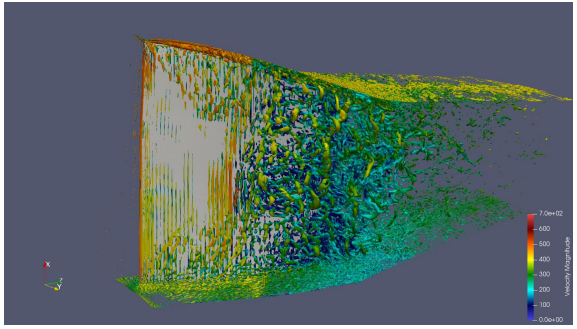
System	Local workstation	Summit supercomputer
CPU/Host	8 core AMD EPYC	42 core IBM POWER9 node
GPU/Device	1 Nvidia P6000	6 Nvidia V100 per node
PGI compiler	19.4	19.9
MPI library	OpenMPI 2.1.6	Spectrum MPI, 10.2.1.2

GPU acceleration of FINE/FR CFD solver with OpenACC

FINE/FR: Representative Trace & Profile



- NASA Rotor37 Test Configuration
- Single passage sector configuration, unsteady simulation with explicit time stepping
- Linux Workstation, 8 core AMD EPYC CPU + NVIDIA P6000 GPU
- Sample-based stack trace from HPCToolkit

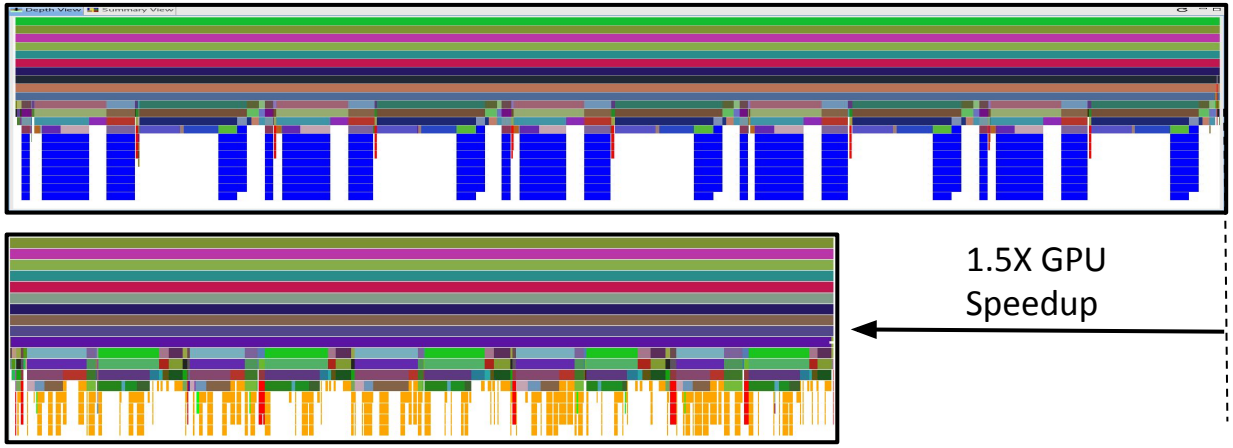


There is no magic bullet, to efficiently leverage GPUs we must offload both BLAS calls and many user routines

GPU acceleration of FINE/FR CFD solver with OpenACC

FINE/FR: Blind Incremental Acceleration

- Incrementally offload the costliest solver hot spots
  - All BLAS calls replaced with cuBLAS
  - Remaining user routines/loops instrumented with OpenACC pragmas
  - Static data (coordinates, constants) offloaded to the device persistently
  - Input/output data to each routine synced conservatively



cuBLAS calls, negligible amount of time

Numerous host/device updates now dominate the execution trace

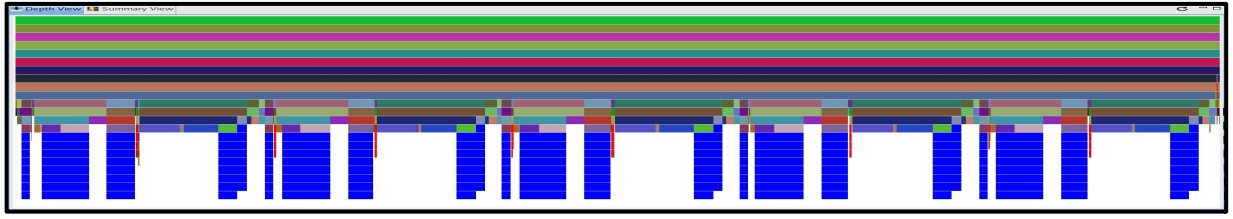
Blind incremental acceleration is easy but insufficient. The host/device transfer of bulk data arrays negates most of the speedup

GPU acceleration of FINE/FR CFD solver with OpenACC

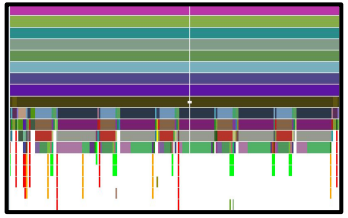
FINE/FR: Minimized Data Transfer

- Host<->Device data transfer must be minimized
  - This is only possible if all operations that touch the bulk data structures are offloaded to the device
  - Some small operations are slower on the device than the host, this is ok
  - All bulk 3D data offloaded to the device permanently
  - The only remaining data transfer that occurs within the time marching loop are for boundary data and infrequent bulk syncs for solution checkpointing to disk

CPU-only
6.2 seconds



CPU+GPU
1.2 seconds



5.1X GPU
Speedup

Host/Device data transfer is no longer a significant consideration.

This can be viewed as an "all-or-nothing" implementation.

GPU acceleration of FINE/FR CFD solver with OpenACC

FINE/FR: Minimized Data Transfer (Continued)

- FINE/FR is a large C++ application with a complicated call tree
  - The “all or nothing” approach to GPU acceleration is not realistic when accelerating a rapidly changing code base with multiple developers
  - A single new routine could lead to data locality bugs elsewhere in the solver that are hard to diagnose
  - With a complicated call tree it is also not always clear when and where data has been modified
  - How do we reduce implementation and maintenance cost?

- Solution: Location-Aware Arrays
  - Wrap all data arrays in a container class
  - In addition to linearized array data and host<->device update methods, this class also contains a “last modified” flag indicating where the array was last updated
  - It is the developer’s responsibility to flag the input/output arrays to all routines, what we refer to as “GPU Boilerplate”
  - This allows the developer to focus on the details of a particular routine, while naturally resulting in minimized data transfer.

- Standalone Example...

GPU acceleration of FINE/FR CFD solver with OpenACC

FINE/FR: Minimized Data Transfer (Continued)

AccArray.H

```
enum AccessType { HOST, DEVICE };

template <typename T>
class AccArray
{
public:
    AccArray(int size);
    ~AccArray();

    #pragma acc routine seq
    T& operator[] (int i)                 { return _data[i];      }
    void setLastAccess(AccessType access) { _lastAccess = access; }
    int  getSize()                        { return _size;         }

    void createDevice();
    void deleteDevice();
    void sync(AccessType access);
    void updateDevice();
    void updateHost();

private:
    T*         _data;
    int        _size;
    AccessType _lastAccess;
};

template <typename T>
AccArray<T>::AccArray(int size)
{
   _data = new T[size];
   _size = size;
}

template <typename T>
AccArray<T>::~AccArray()
{
   delete[] _data;
}
```

```
template <typename T>
void AccArray<T>::createDevice()
{
    #pragma acc enter data copyin(this)
    #pragma acc enter data create(_data[_size])
}

template <typename T>
void AccArray<T>::deleteDevice()
{
    #pragma acc exit data delete(_data[_size])
    #pragma acc exit data delete(this)
}

template <typename T>
void AccArray<T>::updateHost()
{
    #pragma acc update host(_data[_size])
}

template <typename T>
void AccArray<T>::updateDevice()
{
    #pragma acc update device(_data[_size])
}

template <typename T>
void AccArray<T>::sync(AccessType access)
{
    if (_lastAccess != access)
    {
        if (access == HOST)
        {
            updateHost();
        }
        else
        {
            updateDevice();
        }
    }
}
```

Conditional host<->device updates

GPU acceleration of FINE/FR CFD solver with OpenACC

FINE/FR: Minimized Data Transfer (Continued)

main.cpp

```
#include <iostream>
#include <AccArray.h>
using namespace std;

void doubleValues(AccArray<int>& array)
{
    array.sync(DEVICE);
    #pragma acc parallel loop present (array)
    for (int i=0; i<array.getSize(); i++) array[i] = array[i]*2;
    array.setLastAccess(DEVICE);
}

void addValue(AccArray<int>& array, int adder)
{
    array.sync(HOST);
    for (int i=0; i<array.getSize(); i++) array[i] = array[i] + adder;
    array.setLastAccess(HOST);
}

int main(int argc, char** argv)
{
    int size = 10;
    AccArray<int> array(size);
    array.createDevice();

    for (int i=0; i<size; i++) array[i] = i;
    array.setLastAccess(HOST);

    doubleValues(array);
    doubleValues(array);

    array.sync(HOST);
    for (int i=0; i<size; i++)
        cout << "expected, calculated = " << (i*2*2) << "," <<array[i] << endl;
    array.deleteDevice();
}
```

"GPU Boilerplate"

Values set up on the host

Two accelerated functions.

```
:$ pgc++ -ta=tesla -acc -Minfo=acc main.cpp -I ./
doubleValues(AccArray<int> &):
     10, Generating present(array[:])
         Generating Tesla code
         12, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
AccArray<int>::operator [](int):
      4, include "AccArray.h"
          11, Generating acc routine seq
              Generating Tesla code
AccArray<int>::getSize():
      4, include "AccArray.h"
          13, Generating implicit acc routine seq
              Generating acc routine seq
              Generating Tesla code
AccArray<int>::createDevice():
      4, include "AccArray.h"
          45, Generating enter data copyin(this[:1])
              Generating enter data create(_data[:_size])
AccArray<int>::deleteDevice():
      4, include "AccArray.h"
          52, Generating exit data delete(this[:1],_data[:_size])
AccArray<int>::updateDevice():
      4, include "AccArray.h
          64, Generating update device(_data[:_size])
AccArray<int>::updateHost():
      4, include "AccArray.h"
          58, Generating update self(_data[:_size])

:$ ./a.out
expected, calculated = 4,4
expected, calculated = 8,8
expected, calculated = 12,12
expected, calculated = 16,16
expected, calculated = 20,20
expected, calculated = 24,24
expected, calculated = 28,28
expected, calculated = 32,32
expected, calculated = 36,36
expected, calculated = 40,40
```

GPU acceleration of FINE/FR CFD solver with OpenACC

FINE/FR: Minimized Data Transfer (Continued)

main.cpp

```
#include <iostream>
#include <AccArray.h>
using namespace std;

void doubleValues(AccArray<int>& array)
{
    array.sync(DEVICE);
    #pragma acc parallel loop present (array)
    for (int i=0; i<array.getSize(); i++) array[i] = array[i]*2;
    array.setLastAccess(DEVICE);
}

void addValue(AccArray<int>& array, int adder)
{
    array.sync(HOST);
    for (int i=0; i<array.getSize(); i++) array[i] = array[i] + adder;
    array.setLastAccess(HOST);
}

int main(int argc, char** argv)
{
    int size = 10;
    AccArray<int> array(size);
    array.createDevice();

    for (int i=0; i<size; i++) array[i] = i;
    array.setLastAccess(HOST);

    doubleValues(array);
    addValue(array,1);
    doubleValues(array);

    array.sync(HOST);
    for (int i=0; i<size; i++)
        cout << "expected, calculated = " << (i*2+1)*2 << "," <<array[i] << endl;
    array.deleteDevice();
}
```

"GPU Boilerplate"

We add a new function call that modifies the data on the CPU. No changes needed to the host<->device management elsewhere.

```
:$ pgc++ -ta=tesla -acc -Minfo=acc main.cpp -I ./
doubleValues(AccArray<int> &):
     10, Generating present(array[:])
         Generating Tesla code
         12, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
AccArray<int>::operator [](int):
      4, include "AccArray.h"
          11, Generating acc routine seq
              Generating Tesla code
AccArray<int>::getSize():
      4, include "AccArray.h"
          13, Generating implicit acc routine seq
              Generating acc routine seq
              Generating Tesla code
AccArray<int>::createDevice():
      4, include "AccArray.h"
          45, Generating enter data copyin(this[:1])
              Generating enter data create(_data[:_size])
AccArray<int>::deleteDevice():
      4, include "AccArray.h"
          52, Generating exit data delete(this[:1],_data[:_size])
AccArray<int>::updateDevice():
      4, include "AccArray.h"
          64, Generating update device(_data[:_size])
AccArray<int>::updateHost():
      4, include "AccArray.h"
          58, Generating update self(_data[:_size])

:$ ./a.out
expected, calculated = 2,2
expected, calculated = 6,6
expected, calculated = 10,10
expected, calculated = 14,14
expected, calculated = 18,18
expected, calculated = 22,22
expected, calculated = 26,26
expected, calculated = 30,30
expected, calculated = 34,34
expected, calculated = 38,38
```

The results are still correct, at a later date "addValue" could be blindly adapted for GPU acceleration for improved performance.

GPU acceleration of FINE/FR CFD solver with OpenACC

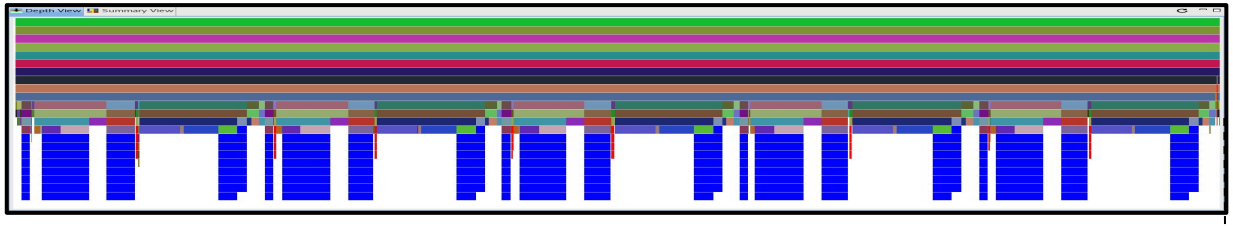
FINE/FR: Minimized Data Transfer (Continued)

- Location Aware Arrays
  - ***Pros:***
    - ***Consistent*** use of location-aware arrays allows the developer(s) to follow a “Blind Incremental Acceleration” approach, naturally yielding an efficient implementation with minimized data transfer
    - New functionality may be implemented on the host with less risk of breaking the existing heterogeneous code.
  - ***Cons:***
    - It may be difficult to retrofit existing data structures, especially AoS data
    - The use of “GPU Boilerplate” is mandatory for this approach to work

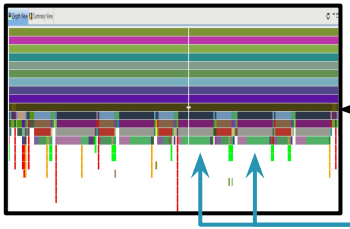
GPU acceleration of FINE/FR CFD solver with OpenACC

FINE/FR: Targeted Optimization

CPU-only
6.2 seconds



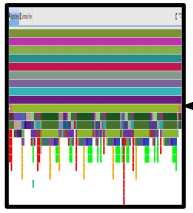
CPU+GPU
1.2 seconds



5.1X GPU
Speedup

Profiling identified a handful of routine with very poor device performance

CPU+GPU
Optimized
0.65 seconds



9.5X GPU
Speedup

Targeted optimization results in an additional 2X performance boost

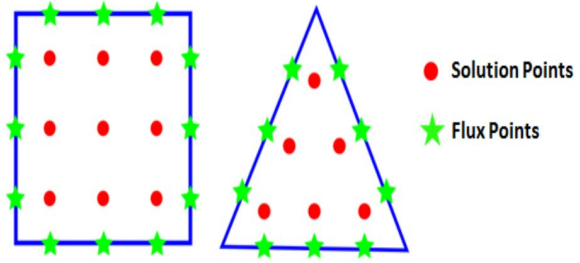
What comes next?

Host/Device data transfer is no longer a major concern.

GPU acceleration of FINE/FR CFD solver with OpenACC

FINE/FR: Targeted Optimization

- The Flux Reconstruction method leads to a pattern of nested loops
  - Outer loop over the faces
  - Inner loop over the number of points per face
  - The number of points per face will vary for different face types and different solution orders
  - Exposed parallelism is limited




```
      #pragma acc parallel loop present(....)
[607] for(int iFace = 0; iFace < nbFaces; iFace++ )
      {
          ...
          int nbPointsFace = getNbPoints(iFace);
          ...
[632]     for (int iPoint = 0; iPoint < nbPointsFace; iPoint++)
          {
              <large amounts of thread safe math>
          }
      }
```

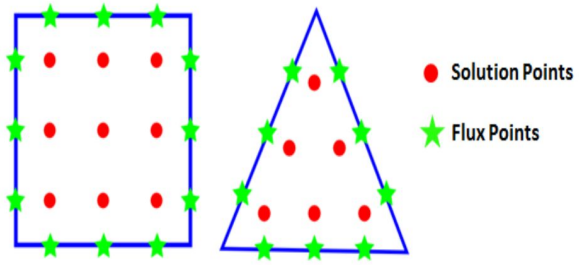
```
PGI 19.4 Compiler, -Minfo=acc output.

Generating Tesla code
607, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
632, #pragma acc loop seq
```

For large partitions there is sufficient parallelism in the outer loop to saturate the device, but for certain operations such as boundary treatment this is not the case.

GPU acceleration of FINE/FR CFD solver with OpenACC

FINE/FR: Targeted Optimization

- Restructuring the loops in a tightly nested form allows the use of the collapse directive for more exposed parallelism at the cost of a number of wasted threads



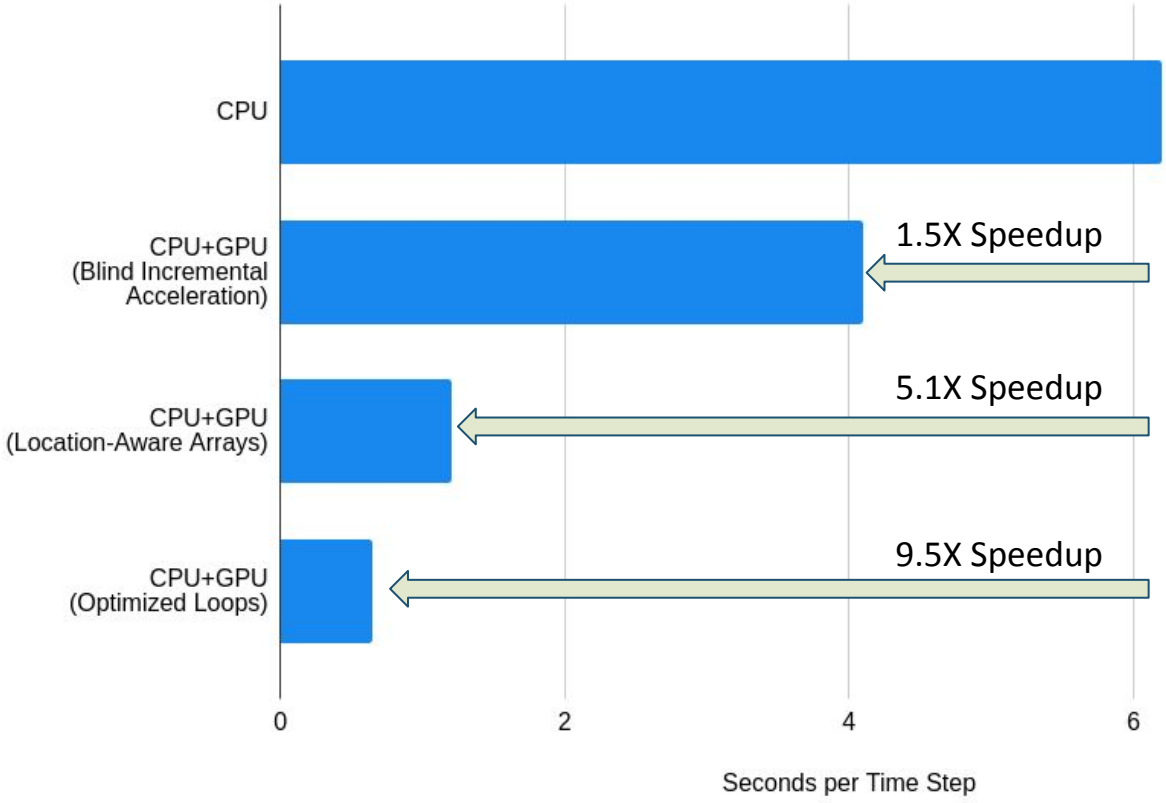

```
     #pragma acc parallel loop collapse(2) present(....)
[607] for(int iFace = 0; iFace < nbFaces; iFace++ )
     {
[609]     for (int iPoint = 0; iPoint < nbPointsFaceMax; iPoint++)
          {
              int nbPointsFace = getNbPoints(iFace);
              if (iPoint < nbPointsFace)
              {
                  <large amounts of thread safe math>
              }
          }
     }
```

```
PGI 19.4 Compiler, -Minfo=acc output.

Generating Tesla code
607, #pragma acc loop gang, vector(128) collapse(2) /* blockIdx.x threadIdx.x
609,   /* blockIdx.x threadIdx.x collapsed */
```

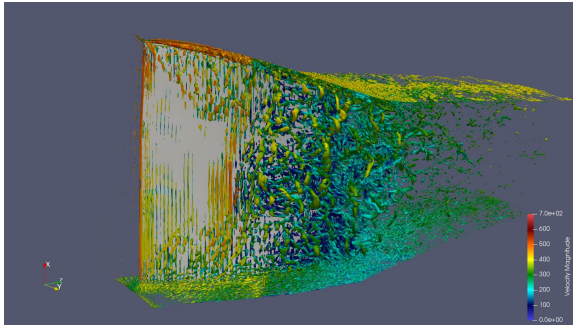
Restructuring of loops for maximum exposed parallelism greatly improves device execution speed. Interestingly, this has been shown to have little negative impact on the CPU performance.

GPU acceleration of FINE/FR CFD solver with OpenACC

FINE/FR: GPU Acceleration



- NASA Rotor37 Test Configuration
- Single passage sector configuration, unsteady simulation with explicit time stepping
- Linux Workstation, 8 core AMD EPYC CPU + NVIDIA P6000 GPU
- Sample-based stack trace from HPCToolkit

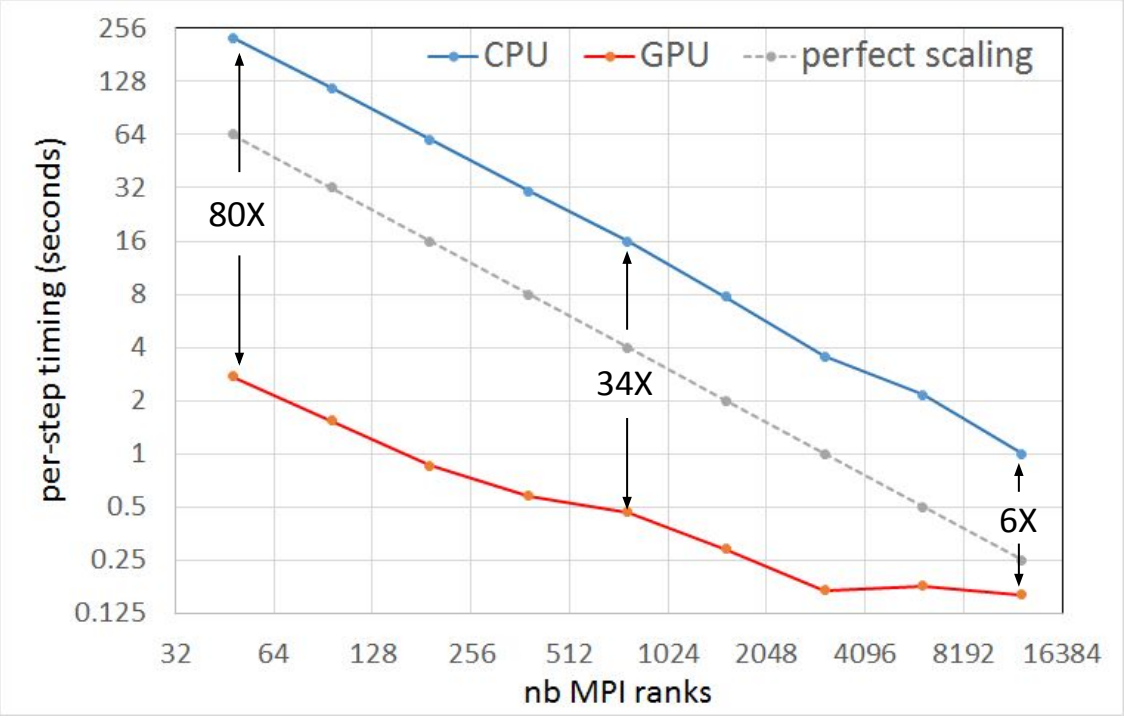


How Does FINE/FR Perform on SUMMIT?

GPU acceleration of FINE/FR CFD solver with OpenACC

FINE/FR: OLCF SUMMIT Scalability

- At-Scale Strong Scalability Demonstration
  - NASA Rotor37, 8M cells, order 3 polynomial flux reconstruction -> 500M Degree of Freedom
  - 1 GPU is paired with 1 CPU (MPI rank), to maximize the partition size at large node count



GPU acceleration of FINE/FR CFD solver with OpenACC

FINE/FR: OLCF SUMMIT Scalability & Discussions

NbNodes	NbCPU & NbGPU	Time (s) CPU	Time (s) CPU+GPU	GPU speedup	NbCell/ Partition	NbDoF/ Partition
8	48	226.00	2.75	82.18	166667	10666667
16	96	117.56	1.54	76.34	83333	5333333
32	192	59.78	0.86	69.51	41667	2666667
64	384	30.59	0.58	52.74	20833	1333333
128	768	15.94	0.47	33.91	10417	666667
256	1536	7.76	0.29	26.76	5208	333333
512	3072	3.57	0.17	20.99	2604	166667
1024	6144	2.15	0.18	11.94	1302	83333
2048	12288	1.00	0.16	6.25	651	41667

- substantial decrease in partition size
  - GPUs not saturated, deviation from linear scalability in the GPU run
- More math favors good GPU speedup: higher orders, larger geometry
- Caution with 80X speedup: cuBLAS highly optimized than netlib BLAS

GPU acceleration of FINE/FR CFD solver with OpenACC

Conclusions & Next Steps

- Strategies for efficient GPU acceleration while maintaining portability and easing long term code maintenance
  - minimized data transfers, via location-aware arrays, “GPU boilerplate”
  - optimization targeted at improving exposed parallelism to GPUs
  - GPU speedup 9.5X on local workstation, 6X - 80X on Summit supercomputer
  - continued optimization to provide sufficient computation to saturate GPUs
  - hopes for future support of virtual functions, C++ vectors etc

Thank you for your (virtual) attention

Questions?

email xiaomeng.zhai@numeca.be

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

