**Michel Müller**
michel@typhooncomputing.com
HPC Programming Framework Research Team, RIKEN

# HYBRID FORTRAN

## New Directive Based GPGPU / CPU Framework - Applied to the Physical Core of the **Weather Simulation Model** ASUCA

## 1. Abstract

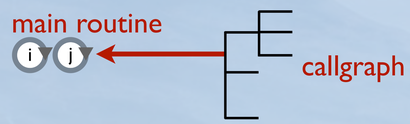
One of the biggest challenges when applying GPGPU frameworks to complex codebases is the necessity for redesigning loops and data access patterns.

Experiences with the Physical Core of the ASUCA weather prediction model have shown that using pure CUDA Fortran or OpenACC leads to a lengthy manual redesign and large execution time overheads when executing the new code back on the CPU.

The Hybrid Fortran meta programming framework has been designed to (a) automate this process and (b) be able to run the user code in CPU optimized loop structure as well, thus enabling optimal performance both on GPU and CPU. Results when using it for the ASUCA Physical Core show High GPU performance, CPU performance on par with the original x86-optimized code, and reduced portation overhead.

## 2. Motivation

The original ASUCA Physical Core is built using a complex callgraph called for every i,j - column - this is a typical CPU optimized structure, where context switch overhead is large and caching is optimized for locality of passed data. No communication in i, j - direction is needed.

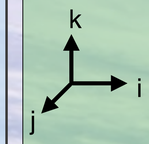



How can we..
- achieve optimal performance on the GPU,
- while not hurting CPU performance too much,
- *and* reduce the amount of critical code changes necessary?

Trying to use OpenACC has shown us that it would lead to
(a) a lengthy manual redesign of the loop structure where the loops are situated closer at the computational code.
(b) a large performance decrease when running this restructured code on the CPU.

The idea of the **Hybrid Fortran** framework is to abstract the loops and automate the restructuring process, such that we can (a) save a lot of development time and (b) have optimal loop structures both for GPU **and** CPU.

## Legend / Geometry




Geometry:
horicontal domains:
i, j
vertical domain:
k


Loop over i, j domains

**Come see my Talk:**

**Wed, 15:30, Room 211B**

**Framework is Open Source!**

- Prepocessor implemented in Python 2.7
- Cross platform capable
- Includes Build- and Test System (based on GNU Make / Bash Script)

**E-Mail me at michel@typhooncomputing.com for GitHub Link / Thesis.**

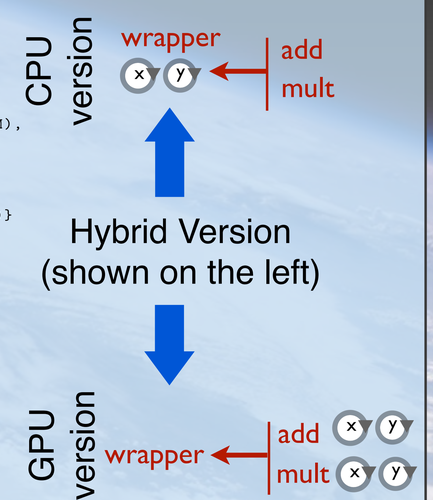
Pull requests are welcome!

Image courtesy of NASA

## 3. Hybrid Fortran Directives (example)

Hybrid Fortran adds two directives to the Fortran language:
1. `@domainDependant` specifies the symbols for each domain (here only x,y,z-domain).
2. `@parallelRegion` is an abstraction of both for-loops and GPU kernels. The `appliesTo` property is used to specify whether this region applies to CPU, GPU or both.

```
1  module example
2  contains
3    subroutine wrapper(a, b, c, d)
4      real, dimension(NZ), intent(in) :: a, b
5      real, dimension(NZ), intent(out) :: c, d
6
7      @domainDependant{domName(x,y,z), domSize(NX,NY,NZ), domPP(DOM),
           accPP(AT)}
8      a, b, c, d
9      @end domainDependant
10
11     @parallelRegion{appliesTo(CPU), domName(x,y), domSize(NX, NY)}
12     call add(a, b, c)
13     call mult(a, b, d)
14     @end parallelRegion
15   end subroutine
17   subroutine add(a, b, c)
18     real, dimension(NZ), intent(in) :: a, b
19     real, dimension(NZ), intent(out) :: c
20     integer :: z
22     @domainDependant{domName(x,y,z), domSize(NX,NY,NZ), domPP(DOM),
           accPP(AT)}
23     a, b, c
24     @end domainDependant
25
26     @parallelRegion{appliesTo(GPU), domName(x,y), domSize(NX, NY)}
27     do z=1,NZ
28       c(z) = a(z) + b(z)
29     end do
30     @end parallelRegion
31   end subroutine
```



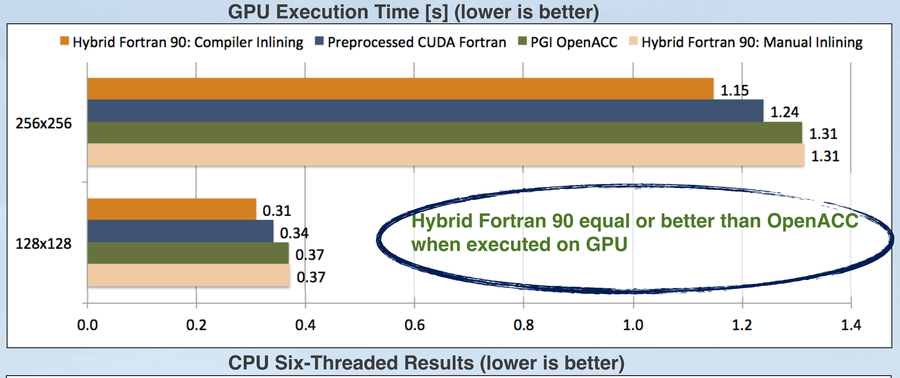

## 4. Performance Results in Comparison with OpenACC and CUDA Fortran

Results from single TSUBAME 2.0 nodes (Tesla M2050 / Xeon X5670)

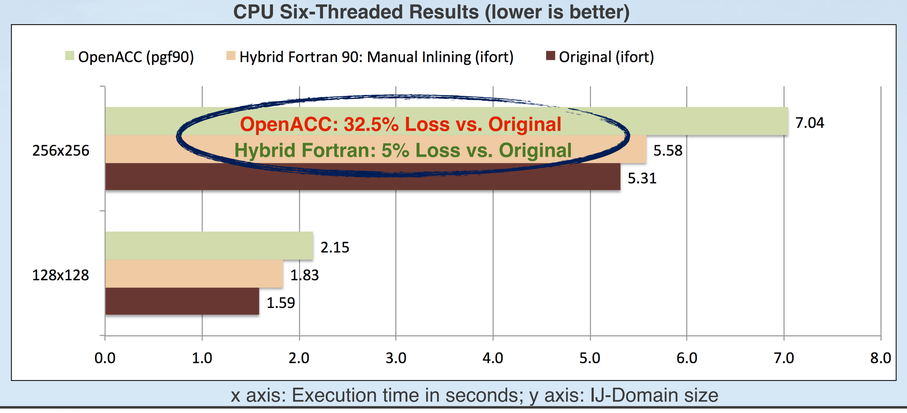
Test Sample: Shortwave Radiation Module in Fortran 90, 4 Kernels, Memory Bounded Problem
All compiler settings have been performance optimized for every individual result.

**HF 90 Compiler Inlining:** Reduced register pressure by grouping some functionality into scalar device subroutines - inlined by compiler
**HF 90 Manual Inlining:** Scalar device functions inlined in global subroutine - matches the OpenACC code version
**Preprocessed CUDA Fortran:** CUDA Fortran reference GPU version
**PGI OpenACC:** OpenACC code with GPU acceleration using the PGI 12.4 implementation
**Original (ifort):** Intel Fortran compiled original ASUCA PP code.
**OpenACC (pgf90):** OpenACC code executed on CPU only, using pgf90 compiler and OpenMP directives. Note: pgf90 was the fastest CPU compiler for the OpenACC code version. ifort was faster for all other code versions.

**GPU Execution Time [s] (lower is better)**




**CPU Six-Threaded Results (lower is better)**



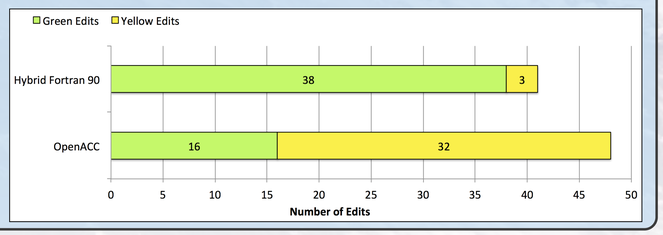

x axis: Execution time in seconds; y axis: IJ-Domain size

## 5. Usability

1.The framework creates intermediate CUDA Fortran / x86 Fortran source code that remains completely human readable (GPU implementation has been designed based on how a CUDA programmer would port the code).

2.A debug GPU implementation has been implemented that automatically prints input / output data for all kernels at a user specified point. It is invoked simply though a Make parameter. Note: CUDA Fortran as well as PGI OpenACC have not yet offered a device debug mode - this functionality has thus been proven tremendously helpful for debugging.

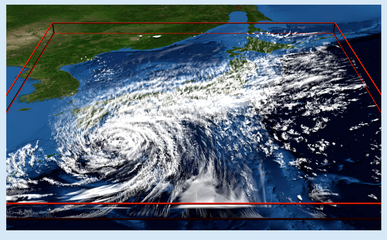
3.Usability study based on a sample kernel typical for the ASUCA physical core. Yellow Edits: Modifications within the computational part; Green Edits: Find/replaceable Modifications / Newly introduced setup code at the beginning and end of a subroutine.




## 6. Conclusion, Current and Future Work

1. A GPGPU strategy has been developed to meet the challenges of an ASUCA Physical Core implementation.
2. Appropriate tools have been invented to make such an implementation possible.
3. Usability and performance of these tools have been validated.

**Current** and **near future** work involves
1. complete portation of ASUCA Physical Core to Hybrid Fortran (under way at RIKEN), joining the already CUDA-ported Dynamical Core - enabling more accurate prediction models through higher performance at the same cost,
2. general Stencil compatibility (because of lack of necessity for ASUCA Physical Core, data accesses with horizontal offsets are currently not implemented - however the framework is prepared for future improvement in that regard),
3. support for more implementations, e.g. OpenCL, Intel MIC.



東京工業大学
Tokyo Institute of Technology

GSIC
Global Scientific Information and Computing Center

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Japan Meteorological Agency

RIKEN

**Supervised by**
Dr. Takashi Shimokawabe, Tokyo Institute of Technology
Prof. Dr. Aoki, Tokyo Institute of Technology
Dr. Tabito Hara, Japan Meteorological Agency