ASUCA on GPU:

Uncompromising Hybrid Port for Physical Core of Japanese Weather Model

Michel Müller
Typhoon Computing, Zurich Switzerland
michel@typhooncomputing.com

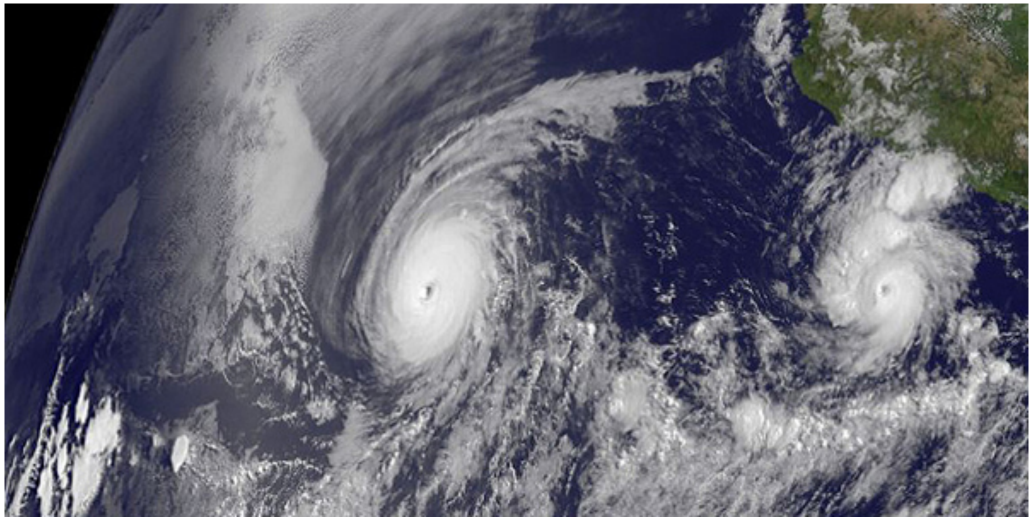
Supervised by

Dr. Naoya Maruyama
RIKEN Advanced Institute for Computational Science

Dr. Tabito Hara
Japan Meteorological Agency

Dr. Takashi Shimokawabe
Tokyo Institute of Technology

Prof. Dr. Takayuki Aoki
Tokyo Institute of Technology



Creative Commons: Nasa Goddard Space Flight Centre, 2010

RIKEN

Japan Meteorological Agency

東京工業大学
Tokyo Institute of Technology

GSIC
Global Scientific Information and Computing Center

ASUCA on GPU:
Uncompromising Hybrid Port for Physical Core of Japanese Weather Model

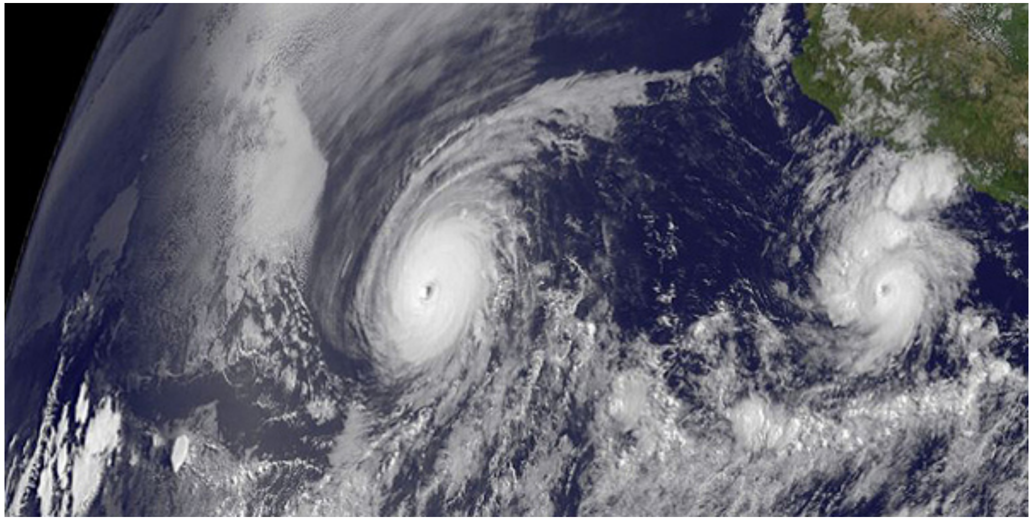
Michel Müller
Typhoon Computing, Zurich Switzerland
michel@typhooncomputing.com

Supervised by
Dr. Naoya Maruyama
RIKEN Advanced Institute for Computational Science

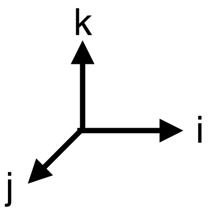
Dr. Tabito Hara
Japan Meteorological Agency

Dr. Takashi Shimokawabe
Tokyo Institute of Technology

Prof. Dr. Takayuki Aoki
Tokyo Institute of Technology



- Goals for ASUCA GPGPU portation of physical core:
  - Eliminate host-to-device communication with GPGPU version of the Dynamical Core
  - Gain execution time speedups
  - Portation must remain CPU compatible
  - Portation must remain performant on CPU

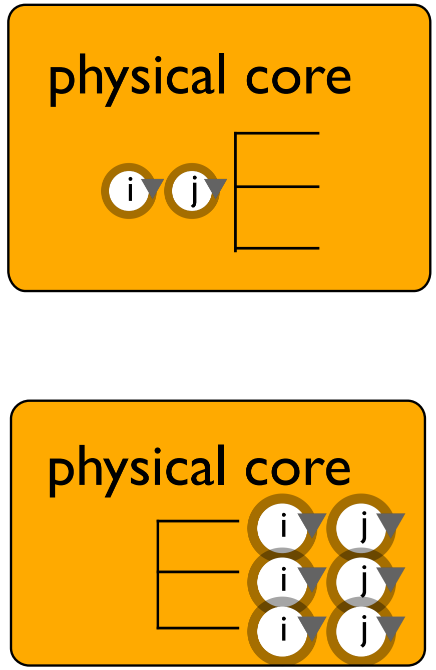


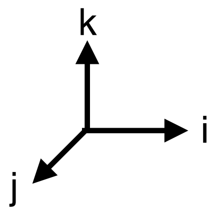
Introduction | Hybrid Fortran

OpenACC approach

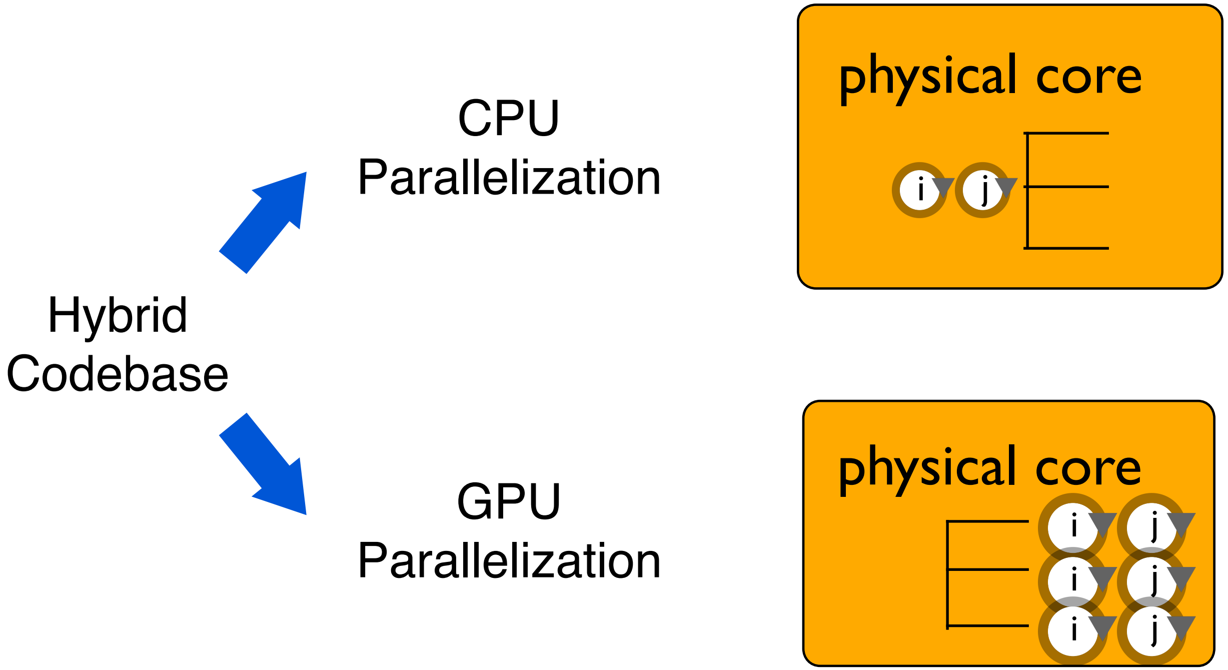
Original (CPU):
Parallelization
at root

GPU
Compatible
Parallelization





Hybrid Fortran approach



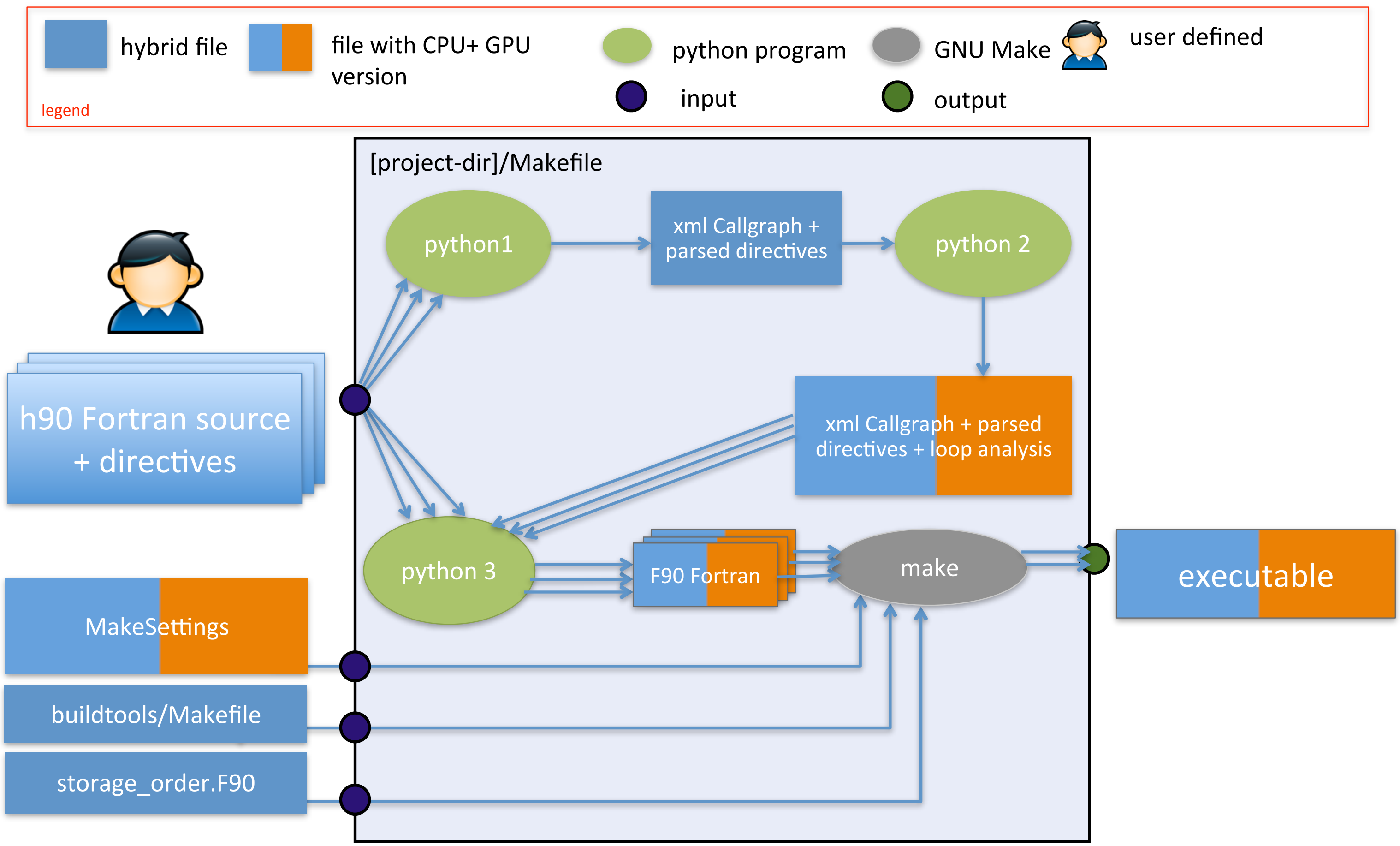
Directives


```fortran
module example
contains
  subroutine wrapper(a, b, c, d)
    real, intent(in) :: a(DOM(NX, NY, NZ)), b(DOM(NX, NY, NZ))
    real, intent(out) :: c(DOM(NX, NY, NZ)), d(DOM(NX, NY, NZ))
    integer(4) :: x, y

    do y=1,NY
    @parallelRegion{appliesTo(CPU), domName(x,y), domSize(NX, NY)}
    do x=1,NX
        call add(a(AT(x,y,:)), b(AT(x,y,:)), c(AT(x,y,:)))
        call mult(a(AT(x,y,:)), b(AT(x,y,:)), d(AT(x,y,:)))
    @end parallelRegion
    end do
    end do
  end subroutine

  subroutine add(a, b, c)
    real, intent(in) :: a(NZ), b(NZ)
    real, intent(out) :: c(NZ)
    integer :: z
    @parallelRegion{appliesTo(GPU), domName(x,y), domSize(NX, NY)}
    do z=1,NZ
      c(z) = a(z) + b(z)
    end do
    @end parallelRegion
  end subroutine
```

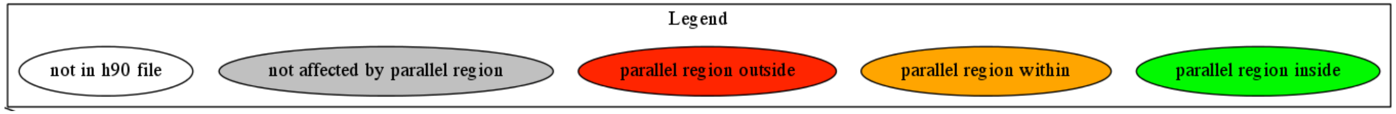
Directives | Hybrid Fortran

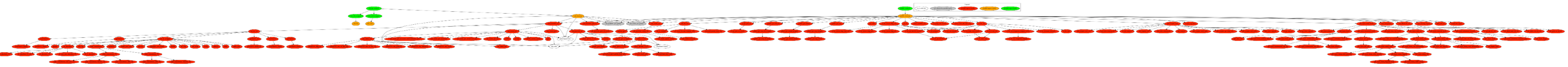
```
module example
contains
    subroutine wrapper(a, b, c, d)
        real, dimension(NZ), intent(in) :: a, b
        real, dimension(NZ), intent(out) :: c, d
        real, dimension(NZ2), intent(out) :: e
        @domainDependant{domName(x,y), domSize(NX, NY), attribute(autoDom)}
        a, b, c, d, e
        @end domainDependant

        @parallelRegion{appliesTo(CPU), domName(x, y), domSize(NX, NY)}
        call add(a, b, c)
        call mult(a, b, d)
        …
        @end parallelRegion
end subroutine
```

Build System | Hybrid Fortran



Scope of ASUCA PP | Implementation

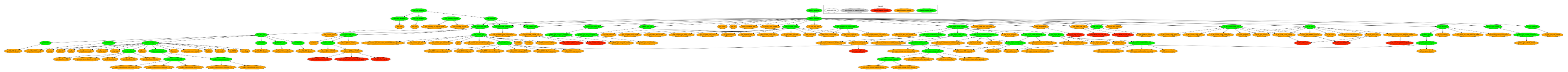




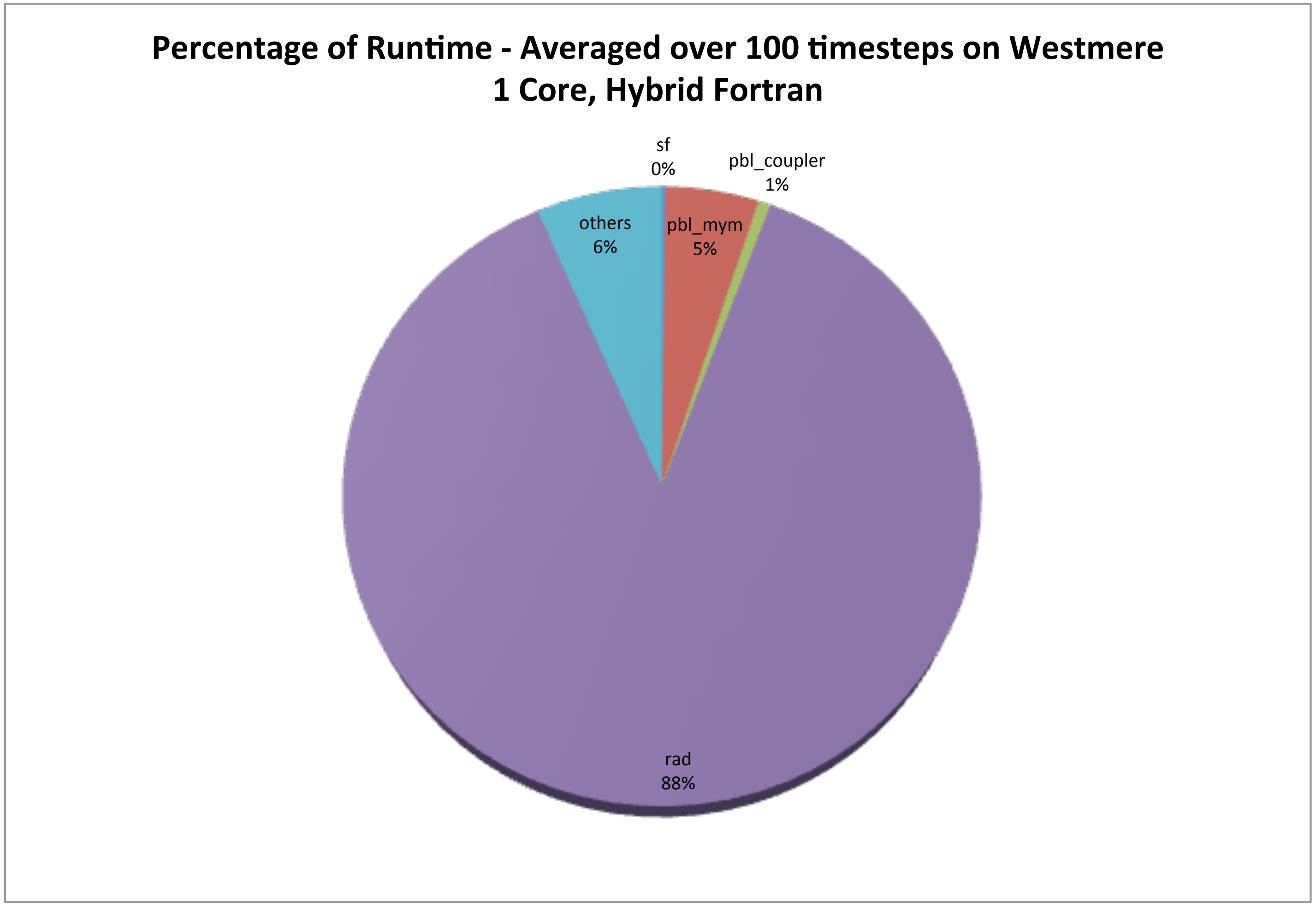
CPU Parallelization

Hybrid

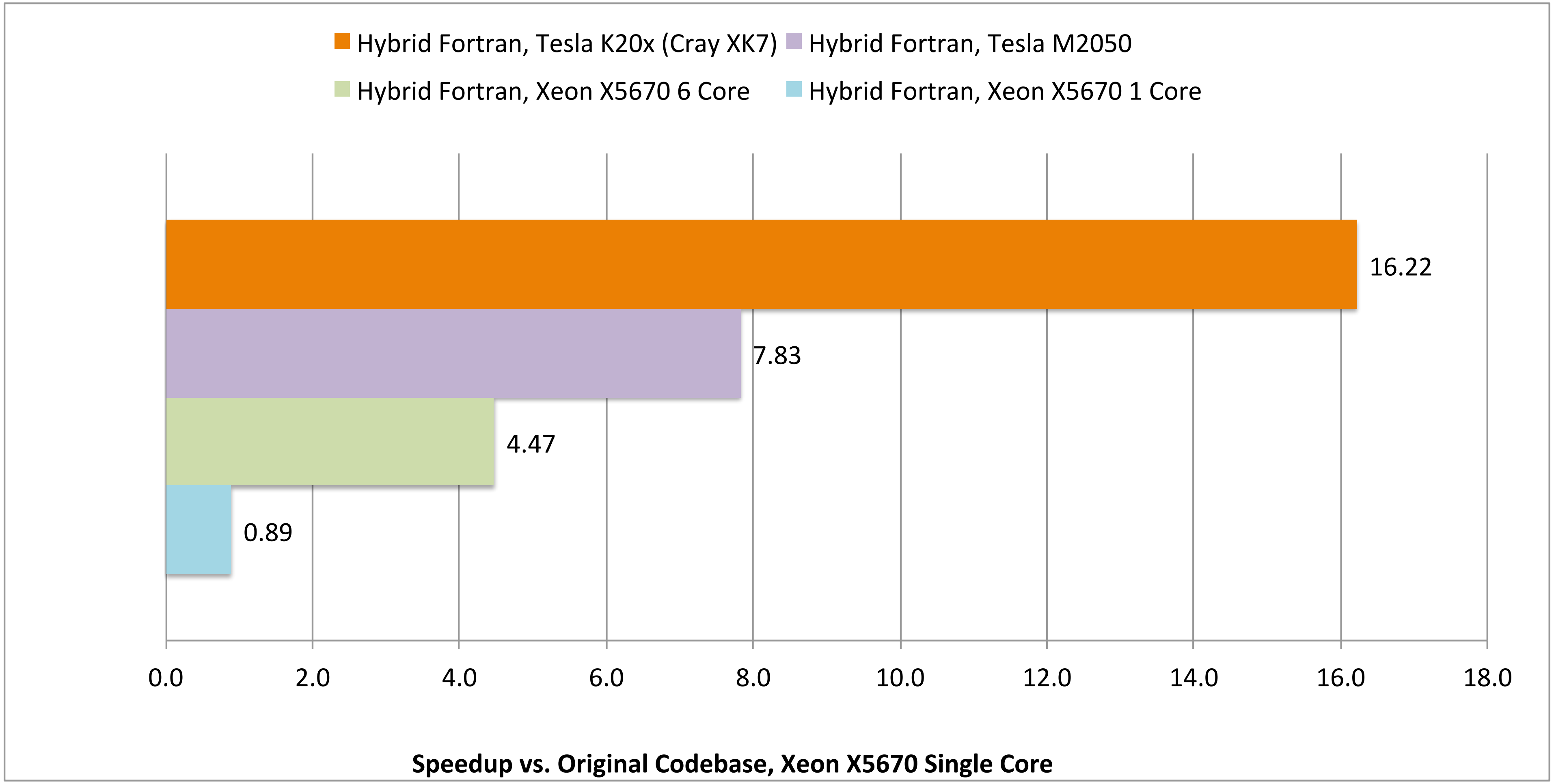
GPU Parallelization



Source Code Demo | Implementation

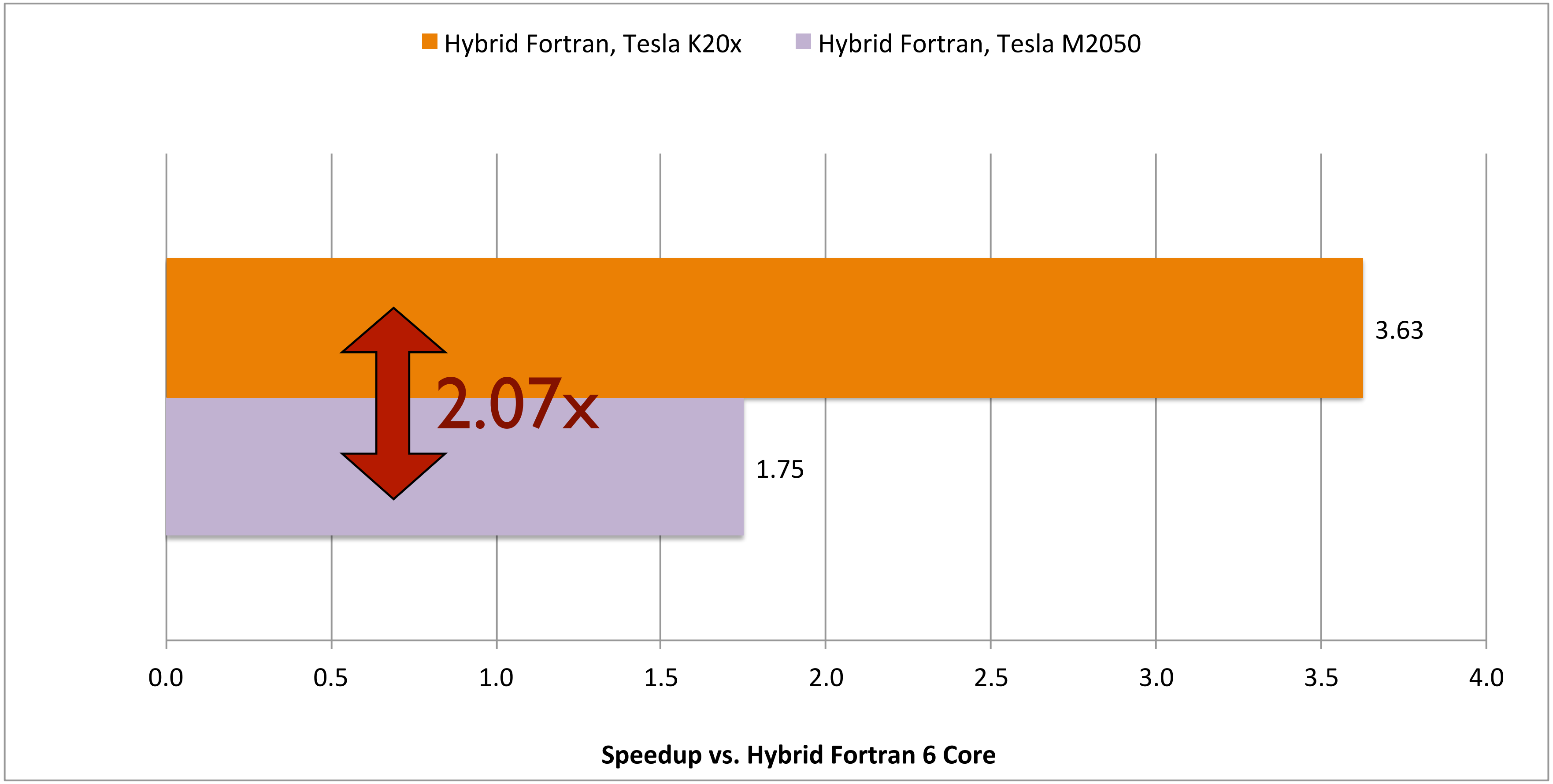

Percentage of Runtime - Averaged over 100 timesteps on Westmere
1 Core, Hybrid Fortran
sf
0%
pbl_coupler
1%
others
6%
pbl_mym
5%
rad
88%

Speedup Gabls3 vs. 1 Core
128 x 128 x 70, 100 Timesteps

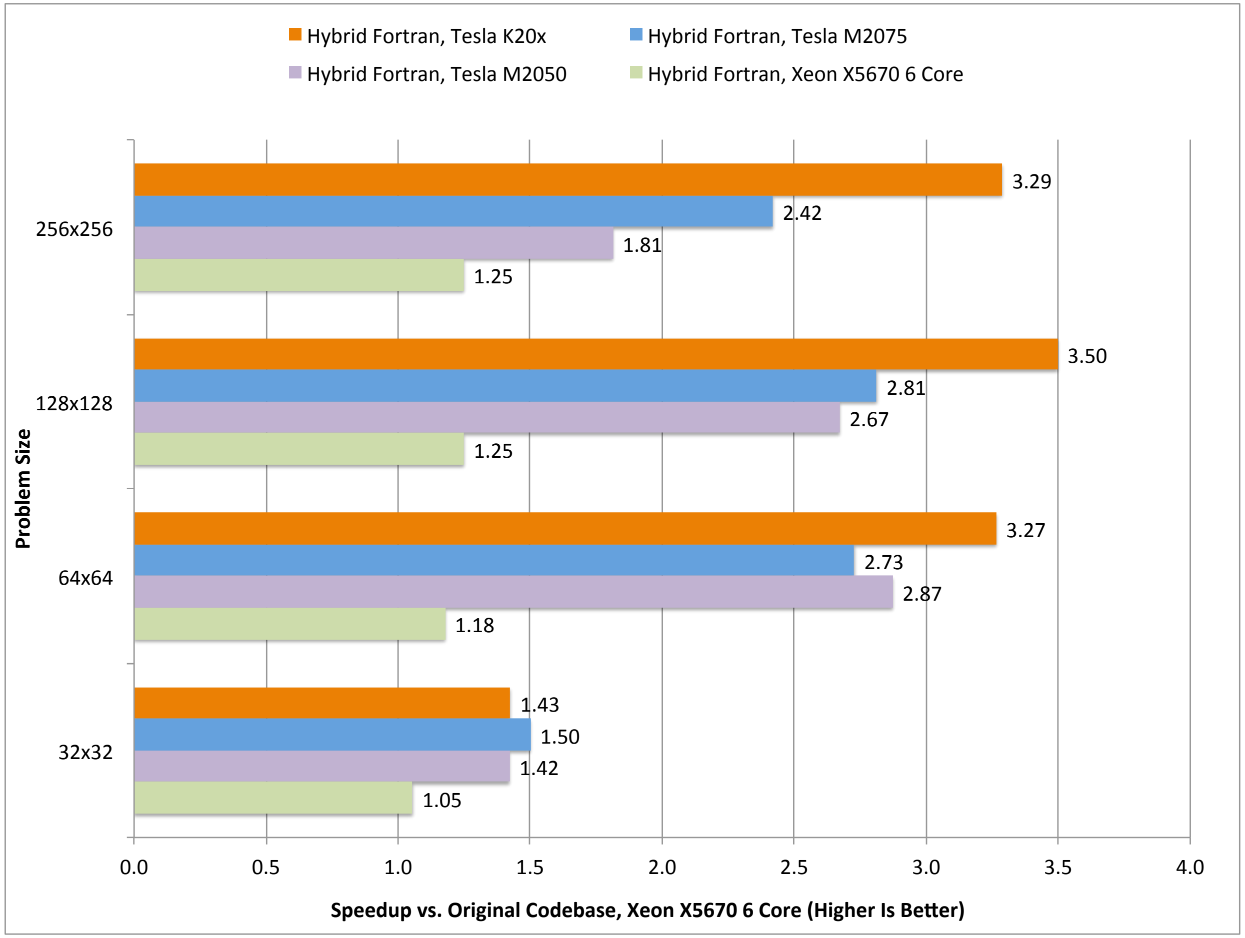


Speedup Gabls3 vs. 6 Core

128 x 128 x 70, 100 Timesteps



Speedup Radiation vs. 6 Core , I x J x 63, Single Timestep



Note: Speedup Single Core Hybrid Fortran vs. Single Core Original: 1-3%

Why is HF Radiation Faster?

- Original Implementation: Transmission Functions used in Longwave Radiation use ~4MB of Temporary Memory per Thread.

- This was too much for GPGPU, where ~10^5 Threads are needed to saturate performance in this case.

- The Transmission therefore needed to be redesigned. Now only uses ~400KB per threads, a 10x improvement.

Why is HF Radiation Faster?

Source Code

Accuracy

after 450 Timesteps on GPU vs. JMA Reference

Results & Discussion

coefccs	1.776357E-15
coefc	1.081428E-14
cov	1.552159E-17
dlwbcs	6.82121E-13
dlwb	6.82121E-13
dswbcs	8.6402E-12
dswb	8.6402E-12
dswt	4.547474E-13
l_mo_inv	2.053913E-15
pt	1.136868E-13
qke	1.777826E-13
qsq	1.959569E-20
qv	7.314694E-17
qv_sfc	6.938894E-18
rh_sfc	0E+00
rlong	5.232738E-17
rnirb	0E+00
rnird	8.171241E-13
rshrt	1.12005E-16
rvisb	2.273737E-13
rvisd	**9.237056E-12**
tcvr	0E+00
tcwc	0E+00
tg	0E+00
tsq	2.097877E-14
ttranscs	3.885781E-16
ttrans	2.629026E-15
u	1.570067E-13
uf	7.494005E-16
ulwtcs	3.410605E-13
ulwt	3.410605E-13
uswbcs	4.902745E-13
uswb	4.902745E-13
uswtcs	6.593837E-12
uswt	6.593837E-12
v	9.878022E-15
wg	0E+00
zmean	4.440892E-16

$$error = \sqrt{\sum_{1}^{n} (x_{HF} - x_r)^2}$$

Programming Time

~ 3M	Implementation of Hybrid Fortran
~ 3M	Implementation of ASUCA Physical Core

1. Hybridized ASUCA Physics with support for Single-GPU and CPU Multicore have been successfully completed.

2. Hybrid Fortran allows us to get GPU compatibility with baseline performance quickly, without compromising CPU performance.

3. We still have all the flexibility for performance tuning.

Questions

michel@typhooncomputing.com