HYBRID FORTRAN

High Performance, Low Friction GPGPU for Weather Prediction

Michel Müller
RIKEN Advanced Institute for Computational Science
michel@typhooncomputing.com

Supervised by
Dr. Takashi Shimokawabe
Tokyo Institute of Technology

Prof. Dr. Takayuki Aoki
Tokyo Institute of Technology

Prof. Dr. Roger Wattenhofer
ETH Zürich

Dr. Tabito Hara
Japan Meteorological Agency

Dr. Naoya Maruyama
RIKEN Advanced Institute for Computational Science

Creative Commons: Nasa Goddard Space Flight Centre, 2010
HYBRID FORTRAN

High Performance, Low Friction GPGPU for Weather Prediction

CPU + GPU executable
1. Motivation
2. Why not OpenACC?
3. Introducing Hybrid Fortran
4. Implementation
5. Results & Discussion
6. Conclusion & Future Work
Motivation

• What is ASUCA? \cite{Kawano2010}
  • Nonhydrostatic weather prediction model
  • Successor of current JMANHM model
  • Dynamical + physical core
• Goals for ASUCA GPGPU portation of physical core:
  1. Eliminate host-to-device communication
  2. Gain execution time speedups

ASUCA: Challenges

**Motivation**

**Challenges:**

1. Keep familiar development environment (Fortran 90)
2. Amount of code changes as small as possible
3. Keep CPU compatibility
4. Keep CPU performance and enable high GPU performance

**Scope of this Project:**

- Concentrate on single GPU / single CPU socket
Why not OpenACC?
1. GPU performance for computationally bounded problems

Particle Push, TSUBAME 2.0

- CUDA C, fast math
- CUDA C
- PGI OpenACC
- HMPP OpenACC

Speedup vs. Single Core CPU, "icc -fast"
1. GPU performance for computationally bounded problems

2. CPU performance

**Shortwave Radiation (8 Kernels), CPU 6 Core, TSUBAME 2.0**

<table>
<thead>
<tr>
<th>Size</th>
<th>OpenACC (pgf90)</th>
<th>Original (ifort)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128x128</td>
<td>2.15</td>
<td>1.59</td>
</tr>
<tr>
<td>256x256</td>
<td>7.04</td>
<td>5.31</td>
</tr>
</tbody>
</table>

**Execution Time [s] (Lower Is Better)**
Why not OpenACC?

1. GPU performance for computationally bounded problems
2. CPU performance
3. Insufficient abstraction
Hybrid Fortran approach

Original (CPU)

GPU optimized

physical core

i, j

physical core

i, j, i, j, i, j
Hybrid Fortran

Hybrid Fortran approach

physical core

Introduction

Hybrid Fortran

Motivation OpenACC Hybrid Fortran Implementation Discussion Conclusion
Example

```
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
      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 do
    end do
  end subroutine

  subroutine add(a, b, c)
    real, intent(in) :: a(NZ), b(NZ)
    real, intent(out) :: c(NZ)
    integer :: z
    do z=1,NZ
      c(z) = a(z) + b(z)
    end do
  end subroutine
```

Hybrid 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)}
        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 subroutine
  subroutine add(a, b, c)
    real, intent(in) :: a(NZ), b(NZ)
    real, intent(out) :: c(NZ)
    integer :: z
    do z=1,NZ
      c(z) = a(z) + b(z)
    end do
  end subroutine
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)}
        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 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
module example
contains
  subroutine wrapper(a, b, c, d)
    real, dimension(NZ), intent(in) :: a, b
    real, dimension(NZ), intent(out) :: c, d
    @domainDependant{domName(x,y,z), domSize(NX, NY, NZ), domPP(DOM), accPP(AT)}
    a, b, c, d
    @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

Implementation
Build System Implementation

**Legend**

- Blue: hybrid file
- Orange: file with CPU+GPU version
- Green: python program
- Gray: GNU Make
- Brown: user defined

**Diagram**

- h90 Fortran source + directives
- MakeSettings
- buildtools/Makefile
- storage_order.F90
- [project-dir]/Makefile
- python1 ➔ xml Callgraph + parsed directives
- python 2 ➔ xml Callgraph + parsed directives + loop analysis
- python 3 ➔ F90 Fortran
- make ➔ executable

**Output**

- buildtools/Makefile
- [project-dir]/Makefile
- python1 ➔ xml Callgraph + parsed directives
- python 2 ➔ xml Callgraph + parsed directives + loop analysis
- python 3 ➔ F90 Fortran
- make ➔ executable

**Input**

- h90 Fortran source + directives
- MakeSettings
- buildtools/Makefile
- storage_order.F90

**Motivation** ➔ OpenACC ➔ Hybrid Fortran ➔ Implementation ➔ Discussion ➔ Conclusion
Implementation of the Build System

- **legend**
  - hybrid file
  - file with CPU+GPU version
  - python program
  - GNU Make
  - user defined

- **[project-dir]/Makefile**
  - python1: xml Callgraph + parsed directives
  - python 2: xml Callgraph + parsed directives + loop analysis
  - python 3: F90 Fortran
  - make: executable

- **dependencies**
  - h90 Fortran source + directives
  - MakeSettings
  - buildtools/Makefile
  - storage_order.F90
  - input
  - output

- **Motivation**
- **OpenACC**
- **Hybrid Fortran**
- **Implementation**
- **Discussion**
- **Conclusion**
Results & Discussion
Scope of Current Implementation

Results & Discussion
Results: Execution Time

- **Results:**
  - **Execution Time [s] (Lower Is Better):**
  - Problem Size:
    - 32x32
    - 64x64
    - 128x128
    - 256x256

- **Comparative Performance:**
  - Hybrid Fortran, Tesla K20x
  - Hybrid Fortran, Tesla M2075
  - Hybrid Fortran, Tesla M2050
  - Hybrid Fortran, Xeon X5670 6 Core
  - Original Codebase, Xeon X5670 6 Core, -fp model precise

---

**Motivation**

**OpenACC**

**Hybrid Fortran**

**Implementation**

**Discussion**

**Conclusion**
Results: Speedup

### Results: Speedup

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Hybrid Fortran, Tesla K20x</th>
<th>Hybrid Fortran, Tesla M2075</th>
<th>Hybrid Fortran, Tesla M2050</th>
<th>Hybrid Fortran, Xeon X5670 6 Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>32x32</td>
<td>1.05</td>
<td>1.43</td>
<td>1.50</td>
<td>1.42</td>
</tr>
<tr>
<td>64x64</td>
<td>1.18</td>
<td>1.43</td>
<td>1.50</td>
<td>1.42</td>
</tr>
<tr>
<td>128x128</td>
<td>1.25</td>
<td>2.67</td>
<td>2.81</td>
<td>2.87</td>
</tr>
<tr>
<td>256x256</td>
<td>1.25</td>
<td>2.42</td>
<td>2.67</td>
<td>2.87</td>
</tr>
</tbody>
</table>

**Speedup vs. Original Codebase, Xeon X5670 6 Core (Higher Is Better)**
### Expected Speedup

<table>
<thead>
<tr>
<th></th>
<th>Tesla M2050</th>
<th>Xeon X5670</th>
<th>Tesla M2050</th>
<th>Xeon X5670</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Memory Bandwidth</strong></td>
<td>11.1 x</td>
<td>5.3 x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bounded Algorithms *</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Computationally</strong></td>
<td>44 x</td>
<td>7.4 x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bounded Algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[2, p. 4], [3], [5]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Tesla sustained bandwidth according to CUDA SDK bandwidth test; CPU bandwidths according to [2, p. 4], [4, p. 7].
Results & Discussion

Tuned Results: GPU

Overall the GPU performance of the compared approaches is relatively similar in this bandwidth limited case. CUDA Fortran implementations (both implemented by hand and implemented automatically by using the new Hybrid Fortran 90 framework) do perform between 5% and 15% better than PGI OpenACC - a similar result has been found for the simple 3D Diffusion test shown in sec. 2.7.2 where PGI OpenACC performed on par with CUDA C as well. For computationally bound problems however we would expect a much more significant advantage for CUDA code versions based on the results of the Particle Push, shown in sec. 2.7.3, where CUDA C was found to perform 2.3× faster than PGI OpenACC.

Figure 5.9: GPU Speedup results for the “radsw” kernel compared to single core CPU (ifort) grouped by grid sizes.

Figure 5.10: GPU Speedup results for the “radsw” kernel compared to six core CPU (ifort) grouped by grid sizes.

Shortwave Radiation (8 Kernels), CPU 6 Core, TSUBAME 2.0
Tuned Results: CPU

Results & Discussion

Shortwave Radiation (8 Kernels), CPU 6 Core, TSUBAME 2.0

![Graph showing execution time results for different grid sizes and simulation methods.]

- **OpenACC (pgf90)**
- **Hybrid Fortran 90: Manual Inlining (ifort)**
- **Original (ifort)**

<table>
<thead>
<tr>
<th>Grid Size</th>
<th>OpenACC (pgf90)</th>
<th>Hybrid Fortran 90: Manual Inlining (ifort)</th>
<th>Original (ifort)</th>
</tr>
</thead>
<tbody>
<tr>
<td>256x256</td>
<td>7.04</td>
<td>5.58</td>
<td>5.31</td>
</tr>
<tr>
<td>128x128</td>
<td>2.15</td>
<td>1.83</td>
<td>1.59</td>
</tr>
</tbody>
</table>

Execution Time [s] (Lower is Better)
Usability Results & Discussion

- **Green Edits**
  - Hybrid Fortran: 30
  - OpenACC: 16

- **Yellow Edits**
  - Hybrid Fortran: 1
  - OpenACC: 32

Number of Edits

---

Motivation | OpenACC | Hybrid Fortran | Implementation | Discussion | Conclusion
Conclusion

1. Dynamic scripting languages can be successfully used to enhance the functionality of a compiled language.

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.
Future Work

• Full scale ASUCA Physical Core implementation
• Pending Improvements for Hybrid Fortran
  1. General Stencil compatibility
  2. Support for more implementations and hardware, e.g. OpenCL, Xeon Phi
  3. Data handling simplifications
References


Next Steps + Questions

• Hybrid Fortran will be available under LGPL license by this weekend!
• E-Mail me for GitHub-link, thesis, slides, further questions..

michel@typhooncomputing.com