Approaches to porting the LMDZ GCM to CUDA Fortran

Sourav Mukherjee
Centre for Atmospheric Sciences
Indian Institute of Technology Delhi
INDIA
Design and Development of a Unified Modeling System for Seamless Weather and Climate predictions of Monsoons

Project Coordinators
Subodh Kumar, PMV Subbarao, OP Sharma

Project funded by
Ministry of Earth Sciences
Government of India
INVESTIGATORS @ IIT Delhi

Subodh Kumar, Huzur Sharan
Department of Computer Science and Engineering

OP Sharma, HC Upadhyaya, Girija Jayaraman, Sagnik Dey
Centre for Atmospheric Sciences

K Sreenadh
Department of Mathematics

PMV Subbarao, P Talukdar, B Premachandran
Department of Mechanical Engineering

BR Chahar
Department of Civil Engineering

KK Pant, Shantanu Roy
Department of Chemical Engineering
About LMDZ

LMDZ is a general circulation model developed since the 70s at the Laboratoire de Météorologie Dynamique (LMD) of CNRS, France. It is implemented in Fortran 77 and 90. The model includes various variants for the Earth and other planets (Mars, Titan, Venus, Exoplanets). The Z in LMDZ stands for zoom.

More information:

http://lmdz.lmd.jussieu.fr/

LMDZ.3.3 source code:

http://web.lmd.jussieu.fr/trac/browser/LMDZ.3.3/trunk
Atmospheric component of the IPSL integrated climate model LMDZ4

LMDZ 3D dynamical core
Finite difference formulation conserving enstrophy and angular momentum

Single-column model
1D monitor for academic or test cases

Several "Physics": LMDZ parametrized physics
- radiation (Fouquart/Morcrette)
- boundary layer (LDM + options)
- convection (Emanuel and Tiedtke)
- clouds (statistical scheme)
- orography (Lott)
- ...

Atmospheric tracers
- Transport by winds Finite volume methods
- Turbulent mixing
- Convective transport

INCA
- Chemistry
- Aerosol microphysics

Oceanic GCM
ORCA LIM
Sea ice

Glaciers

ORCHIDEE
- Soil thermodyn.
- vegetation
- hydrology
- carbon cycle
Outline

Section 1: Manual porting of LMDZ to CUDA Fortran:
   - Progress so far and porting methodology
   - Difficulties with manual porting

Section 2: Automating porting and parallelization:
   - Why automate?
   - What will the automatically generated code look like?
   - How will it be generated?
   - Etc..
Test Platform

Hardware:
- CPU: Intel(R) Xeon(R) CPU E5606 @ 2.13GHz, 8 cores
- GPU: Tesla C2050

Software:
- PGI Fortran compiler 11.4
  - pgfortran 11.4-0 32-bit target on x86-64 Linux -tp nehalem
  - compilation flags for pgfortran: -Mcuda -fast
    ( -fast sets the optimization level to a minimum of 2)
- CUDA toolkit 3.2
Grid dimensions

<table>
<thead>
<tr>
<th>Number of longitudes</th>
<th>nlon or iim</th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of latitudes</td>
<td>nlat or jjm</td>
<td>72</td>
</tr>
<tr>
<td>Number of vertical levels</td>
<td>nlev or llm</td>
<td>19</td>
</tr>
</tbody>
</table>

Although we intend to run the GPU implementation of LMDZ at higher resolutions, the above resolution has been chosen so that we can verify the GPU results with the CPU results.
Section 1:
Manual Porting and Parallelization
Programming of LMDZ

Programming details about the model:

- The organization and structure of subroutines
- Loop varieties
- Data layout in arrays
- Periodicity property
Program organization and structure

Code is organized into 3 directories. Subroutines in

- **dyn3d**
  - implement atmosphere dynamics.
  - implement the interface between dynamics and physics.

- **filtrez**
  - implement FFT. Called by dynamics subroutines.

- **phylmd**
  - implement atmosphere physics.
Dynamics

Major dynamics-related subroutines in \texttt{dyn3d}:

- geopot
- caldyn
- integrd
- pression
- exner\_hyb
- dissip
caldyn
  └─ covcont
     ├─ pression
     └─ massdair
         └─ massbar
             └─ massbarxy
                 └─ flumass
                     └─ dteta1
                         └─ convmas
                             └─ vitvert
                                 └─ tourpot
                                     └─ dudv1
                                         └─ enercin
                                             └─ bernoui
                                                 └─ dudv2
                                                     └─ advect
                                                          └─ convflu
                                                          └─ filtreg

dissip
  └─ gradiv
      └─ nxgrarot
          └─ divgrad
- Small, modular subroutines. 10-20 lines each.
- 3 kinds of loops:
  - `for i = all longitudes, for j = 2 to nlats-1`
  - `for i = all longitudes, for j = 1 and j = nlats`
  - Periodicity fixer:
    `for i = 1 and i = nlon+1 for j = all latitudes`
- Small, modular subroutines. 10-20 lines each.

- 3 kinds of loops:
  - `for i = all longitudes`,
    `for j = 2 to nlats-1`
  - `for i = all longitudes`,
    `for j = 1 and j = nlats`
  - Periodicity fixer:
    `for i = 1 and i = nlon+1`
    `for j = all latitudes`

- These loops themselves occur inside an `l` loop.
Loops are mostly 2 levels deep:

- Iterator $l$ for iterating through different levels.
- Combined iterator $ij$ for iterating through grid points at a particular level.

If there were an iterator $i$ for iterating over longitudes, and an iterator $j$ for iterating over latitudes, then:

$$ij = (j-1) \times (n_{lon}+1) + i$$

number of longitudes plus one
Loops are mostly 2 levels deep:
- Iterator $l$ for iterating through different levels.
- Combined iterator $ij$ for iterating through grid points at a particular level.
- Arrays are mostly 2 dimensional
Manual translation to CUDA Fortran, an example:

```fortran
subroutine flumass
```

```fortran
dudv2  ;bernoui  ;enercin  ;dudv1  ;tourpot  ;covcont
massbar  ;massdair
massbarxy
advect
convmas
vitvert
dteta1
caldyn
convflu
filtreg
pression
massbar
```
Manual translation to CUDA Fortran, first approach

```
DO 5 l = 1,llm
DO 1 ij = iip2,iip1jm
   pbaru( ij,l ) = massebx( ij,l ) * ucont( ij,l )
1 CONTINUE
DO 3 ij = 1,iip1jm
   pbarv( ij,l ) = masseby( ij,l ) * vcont( ij,l )
3 CONTINUE
5 CONTINUE
```

```
attributes(global)
* SUBROUTINE cuda_flumass1
* (massebx,masseby, vcont, ucont, pbaru, pbarv )

REAL,device:: massebx( iip1jmp1,llm ),masseby( iip1jm,llm ),
* vcont( iip1jm,llm ),ucont( iip1jmp1,llm ),pbaru( iip1jmp1,llm ),
* pbarv( iip1jm,llm )

INTEGER  l,ij

ij = (blockIdx%x-1)*16+threadIdx%x
l = (blockIdx%y-1)*16+threadIdx%y
outer:if(l<=lln. and .ij<=iip1jm) then
   if(ij>=iip2) pbaru( ij,l ) = massebx( ij,l ) * ucont( ij,l )
   pbarv( ij,l ) = masseby( ij,l ) * vcont( ij,l )
endif outer

end subroutine cuda_flumass1
```
Manual translation to CUDA Fortran, first approach

```fortran
attributes(global)
* SUBROUTINE cuda_flumass1
* (massebx,masseby, vcont, ucont, pbaru, pbarv )

REAL,device:: massebx( ip1jmp1,llm ),masseby(ip1jm,llm ),
* vcont( ip1jm,llm ),ucont( ip1jmp1,llm ),pbaru( ip1jmp1,llm ),
* pbarv( ip1jm,llm )

INTEGER l,ij

ij = (blockIdx%x-1)*16+threadIdx%x
l = (blockIdx%y-1)*16+threadIdx%y
outer:if(l<=lln.and.ij<=ip1jm)then

if(ij>=ip2) pbaru( ij,l ) = massebx( ij,l ) * ucont( ij,l )
pbarv( ij,l ) = masseby( ij,l ) * vcont( ij,l )

endif outer

end subroutine cuda_flumass1
```

- Beginning of DO loop
- Calculation of iterator variables' values from thread & block indices
- Beginning of IF block with proper conditions

DO 5 l = 1,llm

DO 1 ij = iip2,ip1jm
  pbaru( ij,l ) = massebx( ij,l ) * ucont( ij,l )
  CONTINUE

DO 3 ij = 1,ip1jm
  pbaru( ij,l ) = masseby( ij,l ) * vcont( ij,l )
  CONTINUE

5 CONTINUE
Manual translation to CUDA Fortran, first approach

```fortran
#include "dimensions.h"
#include "paramet.h"
#include "comgeom.h"

REAL massebx( ip1jmp1, lllm ), masseby( ip1jm, lllm ),
* vcont( ip1jm, lllm ), ucont( ip1jmp1, lllm ), pbaru( ip1jmp1, lllm ),
* pbarv( ip1jm, lllm )

REAL apbarun( ilp1 ), apbarus( ilp1 )

REAL sairen, saireun, saires, saireus, ctn, cts, ctn0, cts0
INTEGER l, ij, i

EXTERNAL SSUM
REAL SSUM

attributes(global)
* SUBROUTINE cuda_flumass1
* (massebx, masseby, vcont, ucont, pbaru, pbarv )

REAL, device:: massebx( ip1jmp1, lllm ), masseby( ip1jm, lllm ),
* vcont( ip1jm, lllm ), ucont( ip1jmp1, lllm ), pbaru( ip1jmp1, lllm ),
* pbarv( ip1jm, lllm )

INTEGER l, ij

ij = (blockIdx%x - 1)*16 + threadIdx%x
l = (blockIdx%y - 1)*16 + threadIdx%y

outer: if(l<=lln .and. ij<=ip1jm) then

if(ij>=ip1jm) pbaru( ij, l ) = massebx( ij, l ) * ucont( ij, l )
pbarv( ij, l ) = masseby( ij, l ) * vcont( ij, l )

endif outer

end subroutine cuda_flumass1
```

Variable declarations
flumass.F
Variable declarations
cuda_flumass.F
Manual translation, **first approach**: 

- **Parallelized in the vertical dimension** (\(l\) loop has been parallelized)
- Most of the code, including all the inner DO loops and assignment statements, is exactly the same.
Manual translation, **first approach**:

- Parallelized in the vertical dimension (l loop has been parallelized):
  - separate threads for l loop
  - Most of the code, including all the inner DO loops and assignment statements, is exactly the same.
Manual translation, **first approach**:

- Parallelized in the vertical dimension (l loop has been parallelized):
  - separate threads for l loop

- Most of the code, including all the inner DO loops and assignment statements, is exactly the same.
Values assigned outside the l loop but used inside it have to be necessarily re-computed for each thread.

Manual translation, first approach:

- Most of the code, including all the inner DO loops and assignment statements, is exactly the same.

- The only exception is a couple of assignment statements outside the l loop, which had to be shifted inside the kernel's IF block.
Manual translation, second approach:

- Inner loops also parallelized to the extent possible.
- Assignment statements no longer look identical to those in the original subroutine. The new kernel structure and efficiency reasons have dictated the introduction of new variable names and array subscripts.
Manual translation, **second approach**. The computations related to:

- the parallelizable inner `i` loops are done across the threads of a block.
- the sequential `i` loops are done in the first thread of a block (`tx=1`).
- the north and the south poles are done in separate blocks (`bx=1` and `bx=gridDim%x`).
Manual translation, **second approach**:

- Inner loops also parallelized to the extent possible.
- Assignment statements no longer look identical to those in the original subroutine. The new kernel structure and efficiency reasons have led to the introduction of new variable names and array subscripts.
Manual translation, second approach:

The code related to \textit{flumass} has been combined with other subroutines called by \textit{caldyn}.

Part of a kernel that is a combination of 
\textit{covcont, pression, massdair, massbar, massbarxy, flumass, dteta1},
and several other subroutines.
It is now possible to parse Fortran source code and classify loops as “between poles”, “at poles”, or “fixing periodicity”.
First approach: summary

Objective: maximize code similarity

- Very few code transformations and rearrangements:
  - Variable and array names, dimensions, lengths unchanged.
  - Most array subscripts unchanged.
  - Order of subroutine calls unchanged.
- One or more kernels per subroutine.
- Block size = $\text{dim3}(16,16,1)$ or $\text{dim3}(16,1,1)$, depending on loop depth.
- Grid size depends on loop depth, loop bounds and block size.
Second approach: summary

Objective: **maximize efficiency** by reducing the number of kernel calls

- Extensive code rearrangements and transformations:
  - Several variable/array names, dimensions, lengths changed. New (temporary) variables also introduced.
    - Length of the 1st dimension of an array = $n_{lon} \times n_{lat}$ instead of $(n_{lon}+1) \times n_{lat}$
  - Order of subroutine calls changed.
  - Number of kernels reduced to just 5. Kernels correspond to several subroutines.
- Block size = $\text{dim3}(n_{lon},1,1)$
- Grid size = $\text{dim3}(n_{lat},n_{lev},1)$ or $\text{dim3}(n_{lat},1,1)$, depending on loop depth.
Approach 2: Subroutine rearrangements

geopot

caldyn

integrd

pression

exner_hyb

dissip

cuda_geopot

cuda_caldyn1

cuda_caldyn2_integrd1_pression_exnhyb

cuda_caldyn3_integrd2

cuda_dissip
Approach 2: Subroutine rearrangements

- geopot
- caldyn
- integrd
- pression
- exner_hyb
- dissipation

- cuda_geopot
- cuda_caldyn1
- cuda_caldyn2_integr
- cuda_caldyn3_integr
- cuda_dissip
Approach 2: Subroutine rearrangements
Periodicity: explanation

- To represent a quantity over the grid of longitudes and latitudes as a rectangular array, equality has to be maintained among certain elements of the array.

- This equality is maintained by inserting “periodicity-fixer” DO loops in appropriate places in the code.
Grid of longitudes and latitudes wrapped around Earth
View of the flattened-out grid
On the flat grid, values of a quantity, say potential temperature \( \text{teta} \), for each set of polar points must be the same.
Same colors but of different brightness indicate adjacent points on a latitude.
In many statements in the model, computations at a point on a longitude require values at adjacent longitudes (at the same latitude). So, computations at the **first longitude** require values at the **second longitude** and at the **last longitude**. Similarly, computations at the **last longitude** require values at the **second last longitude** and at the **first longitude**.
To solve this problem while avoiding conditional statements and modulo operations, an extra longitude is defined which is essentially a copy of the first longitude.
Rows of memory cells corresponding to each latitude

first longitude

last longitude
Actually array layout for a given vertical level

longitude 1

latitude 1

latitude 2

latitude nlat

first longitude

last longitude

extra longitude

first longitude

last longitude

extra longitude

first longitude

last longitude

extra longitude

first longitude

last longitude

extra longitude

...
How periodicity is violated

In an assignment statement, the presence of array subscripts:

- \textit{ij} and \textit{ij+1}
- \textit{ij} and \textit{ij-1}

destroys the periodicity of the array in the L.H.S.
How periodicity is violated

In each of the above statements, the computation of \( x(ij) \) requires the value of \( y \) at an adjacent longitude for the same latitude.
How periodicity is violated

When the value of $ij$ corresponds to an 'extra' longitude, i.e. the value of $ij$ is a multiple of $(nlon+1)$ ...

$x(ij) = x(ij) + y(ij+1)$

$x(ij) = x(ij) + y(ij-1)$
How periodicity is violated

When the value of $ij$ corresponds to an 'extra' longitude, i.e. the value of $ij$ is a multiple of $(nlon+1)$

$\begin{align*}
&\Rightarrow x(ij) = x(ij) + y(ij+1) \\
&\Rightarrow x(ij) = x(ij) + y(ij-1)
\end{align*}$

a wrong value gets assigned to the L.H.S. $x(ij)$. 
How periodicity is violated

\( y(ij+1) \)

'meant' to access

\[ \begin{align*}
\text{\textit{actually accesses}}
\end{align*} \]

\[ x(ij) = x(ij) + y(ij+1) \]

Assume \( ij = \text{a multiple of } (nlon+1) \)

\[ x(ij) = x(ij) + y(ij-1) \]

Essentially, the violation happens because the value at a point in an adjacent latitude gets accessed.
How periodicity is restored

\[ x(i_\text{j}) = x(i_\text{j}) + y(i_\text{j}+1) \]

Assume \( i_\text{j} = \text{a multiple of } (n_{\text{lon}}+1) \)

\[ x(i_\text{j}) = x(i_\text{j}) + y(i_\text{j}-1) \]

DO \( i_\text{j} = 1, (n_{\text{lon}}+1)\times n_{\text{lat}}, n_{\text{lon}}+1 \)
\[ x(i_\text{j}+n_{\text{lon}}) = x(i_\text{j}) \]
ENDDO
How periodicity is restored

Wrong value at extra longitude overwritten by correct value copied from first longitude

\[ x(ij) = x(ij) + y(ij+1) \]

Assume \( ij = a \text{ multiple of } (nlon+1) \)

\[ x(ij) = x(ij) + y(ij-1) \]

DO \( ij = 1, (nlon+1)\times nlat, nlon+1 \)

\[ x(ij+nlon) = x(ij) \]

ENDDO
How periodicity is restored

Wrong value at first longitude overwritten by correct value copied from extra longitude

\[ x(ij) = x(ij) + y(ij+1) \]

\[ x(ij) = x(ij) + y(ij-1) \]

*Assume* \( ij = 1 + \text{a multiple of } (nlon+1) \)

DO \( ij = 1, (nlon+1) \times nlat, nlon+1 \)

\[ x(ij) = x(ij+nlon) \]

ENDDO
Approach 1: maintaining periodicity

- **Problem**: Copy from first to extra longitude not possible, because each memory location is serviced by a different thread, and depending on thread block dimensions, might even be associated with different thread blocks.

- **Solution 1**: 

- **Solution 2**: 

Approach 1: maintaining periodicity

- **Problem**: Copy from first to extra longitude not possible, because each memory location is serviced by a different thread, and depending on thread block dimensions, might even be associated with different thread blocks.

- **Solution 1**: In a thread corresponding to an extra longitude, make the R.H.S. equivalent to the R.H.S. of the corresponding first longitude (i.e. the first longitude belonging to the same latitude as the extra longitude).

- **Solution 2**: 
Maintaining Periodicity: Solution 1

\[ x(ij) = z(ij) + y(ij+1) \]

We want the same R.H.S. for the first and the last longitudes.
Maintaining Periodicity: Solution 1

\[ ij1 = ij \]

IF \((\text{mod}(ij, nlon + 1) == 0)\) THEN \(ij = ij - nlon\) ENDIF

\[ x(ij1) = z(ij) + y(ij + 1) \]

Same R.H.S.
Approach 1: Maintaining periodicity

**Problem:** Copy from first to extra longitude not possible, because each memory location is serviced by a different thread, and depending on thread block dimensions, might even be associated with different thread blocks.

**Solution 1:**

**Solution 2:** Do not do any computation in the thread corresponding to the extra longitude. Instead, in the thread corresponding to the first longitude, store the R.H.S. value in a temporary variable and assign it to memory locations for both the first longitude and the corresponding extra longitude.
Maintaining Periodicity: Solution 2

\[ x(ij) = x(ij) + y(ij+1) \]

We want no computation in the thread corresponding to the extra longitude.

We want to do 2 assignments:

\[ x(nlon+1) = x(1) = x(1)+y(1+1) \]

in the thread corresponding to the first longitude.
Maintaining Periodicity: Solution 2

\[
\text{IF}(\text{mod}(ij,nlon+1).\ne.0)\text{THEN} \quad \text{! If } ij \text{ does not corresp to the extra lon} \\
\text{temp} = x(ij) + y(ij+1) \\
x(ij) = \text{temp} \\
\text{IF}(\text{mod}(ij,nlon+1).\text{eq.1})\text{THEN} \quad \text{! If } ij \text{ corresp to the first lon} \\
x(ij+nlon) = \text{temp} \\
\text{ENDIF} \\
\text{ENDIF}
\]

We want no computation in the thread corresponding to the extra longitude.

\[
x(nlon+1) = x(1) = x(1) + y(1+1)
\]

in the thread corresponding to the first longitude.
Approach 2: Maintaining periodicity

- **Problem**: Copy from first to extra longitude not possible, because each memory location is serviced by a different thread, and depending on thread block dimensions, might even be associated with different thread blocks.

- **Solution 3**: Ignore the memory cells corresponding to extra longitudes. Modulo operations are performed on array subscripts to always make them access the correct longitude.
Maintaining Periodicity: Solution 3

\[ x(ij) = x(ij+1) + y(ij-1) \]

threadBlockSize = dim3(nlon,1,1)
Maintaining Periodicity: Solution 3

We want to replace all instances of $ij+1$ with $f^{+1}(ij)$ such that
Maintaining Periodicity: Solution 3

We want to replace all instances of $ij+1$ with $f^{+1}(ij)$ such that
Maintaining Periodicity: Solution 3

We want to replace all instances of $ij+1$ with $f^{+1}(ij)$ such that

Note that the memory cell corresponding to the extra longitude is not accessed at all
Maintaining Periodicity: Solution 3

We want to replace all instances of $ij-1$ with $f^{-1}(ij)$ such that

$$f^{-1}(ij)$$
We want to replace all instances of $ij-1$ with $f^{-1}(ij)$ such that

Note that the memory cell corresponding to the extra longitude is not accessed at all.
Maintaining Periodicity: Solution 3

Kernel configuration:

threadBlockSize = dim3(nlon,1,1)

! nlat is always chosen to be a multiple of 32
blockGridSize   = dim3(nlat,nlev,1)

First few statements in each kernel:

\[
\begin{align*}
l &= blockIdx\%y ; & bx &= blockIdx\%x \\
tx &= threadIdx\%x \\
ij0 &= (bx-1)*nlns &; & ij &= ij0+tx \\
txm1 &= \text{mod}(nlon+tx-2,nlon)+1 ; & ijm1 &= ij0+txm1 \\
txp1 &= \text{mod}(tx,nlon)+1 &; & ijp1 &= ij0+txp1
\end{align*}
\]
Maintaining Periodicity: Solution 3

- **Array length:** The length of the first dimension of a device array is `nlon*nlat` instead of `(nlon+1)*nlat` for better memory alignment.
Speedup
### Approach 1: speedup in individual subroutines

<table>
<thead>
<tr>
<th>subroutine</th>
<th>CPU time</th>
<th>GPU time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>covcont</td>
<td>277</td>
<td>95</td>
<td>2.91</td>
</tr>
<tr>
<td>pression</td>
<td>116</td>
<td>68</td>
<td>1.72</td>
</tr>
<tr>
<td>massdair</td>
<td>209</td>
<td>91</td>
<td>2.29</td>
</tr>
<tr>
<td>massbar</td>
<td>528</td>
<td>101</td>
<td>5.21</td>
</tr>
<tr>
<td>massbarxy</td>
<td>514</td>
<td>95</td>
<td>5.42</td>
</tr>
<tr>
<td>flumass</td>
<td>346</td>
<td>670</td>
<td>0.51</td>
</tr>
<tr>
<td>dteta1</td>
<td>13,931</td>
<td>2,140</td>
<td>6.51</td>
</tr>
<tr>
<td>convmas</td>
<td>27,597</td>
<td>3,705</td>
<td>7.45</td>
</tr>
<tr>
<td>vitvert</td>
<td>152</td>
<td>80</td>
<td>1.90</td>
</tr>
<tr>
<td>tourpot</td>
<td>14,083</td>
<td>1,584</td>
<td>8.89</td>
</tr>
<tr>
<td>dudv1</td>
<td>558</td>
<td>123</td>
<td>4.52</td>
</tr>
<tr>
<td>enercin</td>
<td>775</td>
<td>368</td>
<td>2.10</td>
</tr>
<tr>
<td>bernoui</td>
<td>13,396</td>
<td>1,387</td>
<td>9.66</td>
</tr>
<tr>
<td>dudv2</td>
<td>896</td>
<td>144</td>
<td>6.22</td>
</tr>
<tr>
<td>advect</td>
<td>2,678</td>
<td>829</td>
<td>3.23</td>
</tr>
<tr>
<td>caldyn</td>
<td>67,844</td>
<td>10,629</td>
<td>6.38</td>
</tr>
<tr>
<td>integrd</td>
<td>25,850</td>
<td>622</td>
<td>41.58</td>
</tr>
<tr>
<td>exner_hyb</td>
<td>159,562</td>
<td>1,521</td>
<td>104.89</td>
</tr>
<tr>
<td>geopot</td>
<td>517</td>
<td>138</td>
<td>3.73</td>
</tr>
</tbody>
</table>
Approach 1: speedup in individual subroutines

- covcont: 2.91
- pression: 1.72
- massdair: 2.29
- massbar: 5.21
- massbarxy: 5.42
- flumass: 0.51
- dteta1: 6.51
- convmas: 7.45
- vitvert: 1.9
- tourpot: 8.89
- dudv1: 4.52
- enercin: 2.1
- bernoui: 9.66
- dudv2: 6.22
- advect: 3.23
- caldyn: 6.38
- integrd: 41.58
- exner_hyb: 104.89
- geopot: 3.73
## Approach 1: speedup in model

<table>
<thead>
<tr>
<th>kernel name</th>
<th>comments</th>
<th>CPU</th>
<th>GPU</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMDZ dynamics (excluding dissip and physics)</td>
<td>Iterations = 400</td>
<td>23,174,576</td>
<td>w/o d.t.</td>
<td>5.50</td>
</tr>
<tr>
<td></td>
<td>Time step = 180s</td>
<td>(= 2.07 minutes)</td>
<td>4,221,719</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Simulated time = 20 hrs</td>
<td></td>
<td>with d.t.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4,234,293</td>
<td>5.47</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>d.t. 12,574</td>
<td></td>
</tr>
</tbody>
</table>
## Approach 2: speedup in combined subroutines

<table>
<thead>
<tr>
<th>Kernel</th>
<th>CPU time</th>
<th>GPU time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>cudacopy_vcov_ucov_teta_masse_ps</td>
<td>984</td>
<td>129</td>
<td>7.63</td>
</tr>
<tr>
<td>cuda_geopot</td>
<td>687</td>
<td>91</td>
<td>7.55</td>
</tr>
<tr>
<td>cuda_caldyn1</td>
<td>25784</td>
<td>666</td>
<td>38.71</td>
</tr>
<tr>
<td>cuda_caldyn2_integrd1_pression_exnerhyb</td>
<td>18537</td>
<td>287</td>
<td>64.59</td>
</tr>
<tr>
<td>cuda_caldyn3_integrd2</td>
<td>10508</td>
<td>373</td>
<td>28.17</td>
</tr>
<tr>
<td>cuda_dissip</td>
<td>39216</td>
<td>2846</td>
<td>13.79</td>
</tr>
</tbody>
</table>

Time in microseconds.
Approach 2: speedup in combined subroutines

<table>
<thead>
<tr>
<th>Function</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>cudacopy_vcov_ucov_teta_masse_ps</td>
<td>7.63</td>
</tr>
<tr>
<td>cuda_geopot</td>
<td>7.55</td>
</tr>
<tr>
<td>cuda_caldyn1</td>
<td>38.71</td>
</tr>
<tr>
<td>cuda_caldyn2_integrd1_pression_exnerhyb</td>
<td>64.59</td>
</tr>
<tr>
<td>cuda_caldyn3_integrd2</td>
<td>28.17</td>
</tr>
<tr>
<td>cuda_dissip</td>
<td>13.79</td>
</tr>
</tbody>
</table>
### Approach 2: speedup in model

<table>
<thead>
<tr>
<th>kernel name</th>
<th>comments</th>
<th>CPU</th>
<th>GPU</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMDZ dynamics</td>
<td>Iterations = 1920</td>
<td>124,403,900</td>
<td>w/o d.t.</td>
<td>42.86</td>
</tr>
<tr>
<td>(including dissip, excluding physics)</td>
<td>Time step = 180s</td>
<td>(= 2.07 minutes)</td>
<td>2,902,558</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Simulated time = 4 days</td>
<td></td>
<td>with d.t.</td>
<td>42.44</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2,931,443</td>
<td></td>
</tr>
<tr>
<td></td>
<td>d.t. 28,585</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Verification of individual kernels

We construct a “driver” program for each kernel and its corresponding subroutines.

- Initialize input variables/arrays with random numbers
- Call subroutine
- Transfer input data to GPU
- Call kernel
- Transfer GPU output to a temporary array (suffixed _ans)
- Compare output of subroutine and kernel (stored in the _ans array)
SUBROUTINE add(x,z)
IMPLICIT NONE
#include "params.h"
#include "common.h"

REAL::x(n),z(n)
INTEGER i

DO i=1,n
    z(i) = x(i)+y(i)
ENDDO

END SUBROUTINE add

INTEGER,PARAMETER::n=1000

REAL::y
COMMON y(n)

filename: add.F90

module add_mod
IMPLICIT NONE
#include "params.h"

REAL,ALLOCATABLE,DEVICE:::y(:)

ATTRIBUTES(GLOBAL) &
SUBROUTINE cuda_add(x,z)
REAL,DEVICE::x(n),z(n)
INTEGER:: i

i=(blockidx%x-1)*tpb + threadIdx%x
IF (i.le.n) THEN
    z(i) = x(i)+y(i)
ENDIF

END SUBROUTINE cuda_add

END MODULE add_mod

filename: cuda_add.F90

filename: params.h

filename: common.h
PROGRAM main
USE cudafor
USE add_mod, ONLY: y_dev => y,&
cuda_add=>cuda_add
#include "params.h"
#include "common.h"

REAL::x(n),z(n),z_ans(n)
REAL,DEVICE::x_dev(n),z_dev(n)

TYPE(dim3)::dimblock,dimgrid
INTEGER,PARAMETER::tpb = 16
INTEGER:: c1,c2,c3,c4,i
REAL:: cputime

CALL random_number(x)
CALL random_number(y)
CALL system_clock(c1)
CALL add(x,z)
CALL system_clock(c2)
cputime=c2-c1

dimblock=dim3(tpb,1,1)
dimgrid =dim3(((n+tpb-1)/tpb,1,1)

CALL system_clock(c1)
CALL system_clock(c1)
ALLOCATE(y_dev(n))
x_dev = x
y_dev = y
CALL system_clock(c3)
CALL cuda_add <<<dimblock,dimgrid>>> (x_dev,z_dev)
i = cudathreadsynchronize()
CALL system_clock(c4)

z_ans = z_dev
CALL system_clock(c2)

errors=0
DO i=1,n
  IF (abs(z_ans(i)-z(i))>1e-5) THEN
    errors=errors+1
  ENDF
ENDDO

PRINT*,'cputime',cputime
PRINT*,'gputime with d.t.',c2-c1,'. Speedup =',cputime/(c2-c1)
PRINT*,'gputime w/o  d.t.',c3-c4,'. Speedup =',cputime/(c3-c4)
PRINT*,'number of mismatches',errors

END PROGRAM main

filename: main_add.F90
Verification method: Shortcomings

By studying the array indices where there are mismatches between the CPU and GPU result-arrays, it is sometimes possible to infer the location of the mistake in the GPU kernel. However, this mistake-tracking method is dependent on the nature of the inputs, and does not guarantee correctness for all possible inputs.
Verification method: Shortcomings

By studying the array indices where there are mismatches between the CPU and GPU result-arrays, it is sometimes possible to infer the location of the mistake in the GPU kernel.

```
DO  ij=1,(nlon+1)*nlat-1
  y(ij)=x(ij+1)
ENDDO
DO  ij=1,(nlon+1)*nlat,nlon+1
  y(ij+nlon)=y(ij)
ENDDO
```

**CPU**

```
ij=(blockidx%x-1)*16+threadidx%x
```

**GPU**

```
IF(ij.LE.(nlon+1)*nlat-1) THEN
  ij1=ij
  IF(mod(ij,nlon+1).EQ.0) THEN
    ij1=ij-iim
  ENDIF
  y(ij1)=x(ij+1)
ENDIF
```
Verification method: Shortcomings

By studying the array indices where there are mismatches between the CPU and GPU result-arrays, it is sometimes possible to infer the location of the mistake in the GPU kernel.

```
DO ij=1,(nlon+1)*nlat-1
  y(ij)=x(ij+1)
ENDDO
DO ij=1,(nlon+1)*nlat,nlon+1
  y(ij+nlon)=y(ij)
ENDDO
```

```
CPU
DO ij=1,(nlon+1)*nlat-1
  y(ij)=x(ij+1)
ENDDO
DO ij=1,(nlon+1)*nlat,nlon+1
  y(ij+nlon)=y(ij)
ENDDO
```

```
GPU
ij=(blockidx%x-1)*16+threadidx%x
IF(ij.LE.(nlon+1)*nlat-1) THEN
  ij1=ij
  IF(mod(ij,nlon+1).EQ.0) THEN
    ij1=ij-iim
  ENDIF
  y(ij)=x(ij+1)
ENDIF
```
Verification method: Shortcomings

By studying the array indices where there are mismatches between the CPU and GPU result-arrays, it is sometimes possible to infer the location of the mistake in the GPU kernel.

```
call random_number(x)
```

Input to \( x \)

Indices of \( y \) where there is mismatch

- Error at \( ij = 97 \)
- Error at \( ij = 194 \)
- Error at \( ij = 291 \)
- Error at \( ij = 388 \)
- Error at \( ij = 485 \)
- ...
- ...
- Error at \( ij = 6984 \)
Verification method: Shortcomings

However, this mistake-tracking method is dependent on the nature of the inputs, and does not guarantee correctness for all possible inputs.

\[ x(:, :) = 1.0 \]

Indices of y where there is mismatch

No errors
Verification of GPU model

After a CPU model run and a GPU model run, we use *Grads* to plot the difference between the CPU and GPU results as an error plot.
Verification of GPU model

Contouring: -30 to 40 interval 10
latitude -90° to 90°
longitude -180° to 180°
(CPU result)

u1 denotes value of zonal wind velocity at the end of the simulation
at z = 5, pressure is 847.769 millibars

Iterations = 1920
Timestep = 180s
Simulated time = 4 days
Verification of GPU model

Latitude -90° to 90°
Longitude -180° to 180°
(GPU LMDZ, approach 2)
Verification of GPU model

Contouring: -0.008 to 0.006 interval 0.001

latitude -90° to 90°
longitude -180° to 180°
(GPU LMDZ, approach 2)
**Physics**

*phylmd* contains subroutines that implement the physics of the model

- Called every 10 steps, consumes more than 20% of the total runtime.
- Certain subroutines are extremely long to be ported manually, for e.g. `radlwsw.F` (6144 lines), `physiq.F` (2710 lines), `clmain.F` (1733 lines).
<table>
<thead>
<tr>
<th></th>
<th>Name of subroutine</th>
<th>Outermost DO loop number in the subroutine</th>
<th>Total number of assignment statements in the DO loop, including those in inner loops</th>
<th>Total number of variable references, excluding variables appearing as array subscripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>radlwsw</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2</td>
<td>46</td>
<td>103</td>
</tr>
<tr>
<td>3</td>
<td>sw</td>
<td>1</td>
<td>5</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>2</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>swu</td>
<td>1</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>2</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>3</td>
<td>30</td>
<td>92</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>6</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>7</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>sw1s</td>
<td>1</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>2</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>3</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>4</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>5</td>
<td>8</td>
<td>28</td>
</tr>
<tr>
<td>Name of subroutine</td>
<td>Outermost DO loop number in the subroutine</td>
<td>Total number of assignment statements in the DO loop, including those in inner loops</td>
<td>Total number of variable references, excluding variables appearing as array subscripts</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>phylmd/radlwsw.F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lwvd</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>14</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>15</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>lwvn</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>21</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>lwtt</td>
<td>1</td>
<td>4</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>52</td>
<td>142</td>
<td></td>
</tr>
<tr>
<td>lwttm</td>
<td>1</td>
<td>4</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>52</td>
<td>157</td>
<td></td>
</tr>
<tr>
<td>Total 21</td>
<td>86</td>
<td>832</td>
<td>2725</td>
<td></td>
</tr>
<tr>
<td>Subroutine</td>
<td>DO LOOP#</td>
<td>Assign Statements</td>
<td>Variable References</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>-------------------</td>
<td>--------------------</td>
<td></td>
</tr>
<tr>
<td>lwvd</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>14</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>15</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>lwvn</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>21</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>lwtt</td>
<td>1</td>
<td>4</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>52</td>
<td>142</td>
<td></td>
</tr>
<tr>
<td>lwttm</td>
<td>1</td>
<td>4</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>52</td>
<td>157</td>
<td></td>
</tr>
</tbody>
</table>

1. Total number of subroutines = 21
2. Total number of outer DO loops = 86
3. Total number of assignment statements inside DO loops = 832
4. Total number of variable references in DO loops, excluding array subscripts = 2725
### phylmd/radlwsw.F

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>lwvd</td>
<td>DO LOOP#</td>
<td>ASSIGN STMTS</td>
<td>VAR REFS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>47</td>
<td>47</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>50</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lwvn</td>
<td>DO LOOP#</td>
<td>ASSIGN STMTS</td>
<td>VAR REFS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>21</td>
<td>88</td>
<td>88</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>7</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lwtt</td>
<td>DO LOOP#</td>
<td>ASSIGN STMTS</td>
<td>VAR REFS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>14</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>52</td>
<td>142</td>
<td>142</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lwttm</td>
<td>DO LOOP#</td>
<td>ASSIGN STMTS</td>
<td>VAR REFS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>15</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>52</td>
<td>157</td>
<td>157</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total 21</td>
<td></td>
<td>86</td>
<td>832</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*These statistics have been automatically generated, not manually counted!*
It is extremely error prone to manually port this code to CUDA, though it has been attempted, but the kernel is generating NaN results.
Section 2: Automating Porting and Parallelization
Why, What, How.

- Why automate?
- What needs to be automated.
- Target language and code structure.
- Theory, techniques, tools.
Why automate?

- To eliminate programmer errors.
- To keep ported/parallelized version in-sync with new versions of the model.
Why automate?

- To eliminate programmer errors
  - Forget initializing an input kernel variable/array
    → errors in output
  - Forget allocating a CUDA array
    → segmentation fault
  - Pass insufficient number of arguments to a subroutine
    → segmentation fault
  - Pass a device variable to a normal subroutine
    → segmentation fault
  - Forget a synchronization statement
    → occasional errors in output!
Why automate?

- To eliminate programmer errors
- To keep ported/parallelized version in-sync with new versions of the model
  - “Approach 2” kernels very inflexible.
  - Modifying them only little better than writing from scratch.
Why, What, How.

- Why automate?
- What needs to be automated.
- Target language and code structure.
- Theory, techniques, tools.
Why, What, How.

- Why automate?
- What needs to be automated.
- Target language and code structure.
- Theory, techniques, tools.

What is possible to automate now!
What needs to be automated

- Determination of the CALL tree for a subroutine. Uses:
  - To inline subroutine calls. Necessary for Approach 2 kernels.
- Categorization of variables appearing in the subroutine as read, written, common, dummy, local, iterator, save.
- Automatic generation of “driver” programs for comparing a subroutine and its corresponding kernel.
What needs to be automated

- Determination of the CALL tree for a subroutine.

- Categorization of variables appearing in the subroutine as read, written, common, dummy, local, iterator, save. Uses:
  - To generate the statements for Host to Device and/or Device to Host data transfers.

- Automatic generation of “driver” programs for comparing a subroutine and its corresponding kernel.
What needs to be automated

- Determination of the CALL tree for a subroutine.
- Categorization of variables appearing in the subroutine as read, written, common, dummy, local, iterator, save.
- Automatic generation of “driver” programs for comparing a subroutine and its corresponding kernel.
How to automate?

The most important requirement for the tasks that have so far been mentioned is a Fortran parser.

I am using the front-end of G95\(^1\) to parse Fortran.

1: www.g95.org
How to automate?

But a parser is not enough. While it is easy to identify the proper places in the source code of g95 where `printf` statements can be inserted to extract particular information about a Fortran program – for e.g., to print the line numbers of the beginning and end of a Fortran subroutine – having to insert `printf` statements and then re-compile the source code of g95 every time we want a new sort of information about a Fortran program is quite inconvenient.
How to automate?

Preferably, there should be a more “accessible” representation of a Fortran program, containing all the syntactical information about the program in “one place”.

Besides, development in C is usually less rapid and flexible than in an interpreted language like Python.
How to automate?

Using an XML representation of the AST
How to automate?

XML has been explored for more than a decade\(^1\) as a possible “universal” representation of the Abstract Syntax Tree (AST) generated by the compiler of a programming language.

Given the wide range of tools available to parse, query and transform XML, it seems like a good choice\(^2\).

G95 has a function to dump a textual representation of the AST of Fortran. I have modified this function to dump an XML representation of the AST.

1: Using XML as an Intermediate Form for Compiler Development (2001), by Roy Germon
2: Re: xml as intermediate representation, [http://compilers.iecc.com/comparch/article/05-09-113](http://compilers.iecc.com/comparch/article/05-09-113)
How to automate?

Inspiration of the idea came from:


G95-XML is an earlier attempt to dump the Fortran AST as XML. See:

http://g95-xml.sourceforge.net/

Comp.compilers discussion thread:

http://compilers.iecc.com/comparch/article/05-09-113
Solution to the “parsing” challenge

Many options are available to query and transform XML. I am using the `xml.etree` library in Python to query and transform the XML dump of the Fortran AST.
Generating statistics for `phylmd/radlwsw.F`

<table>
<thead>
<tr>
<th>lwvd</th>
<th>DO LOOP#</th>
<th>ASSIGN STMTS</th>
<th>VAR REFS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>14</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>15</td>
<td>50</td>
</tr>
<tr>
<td>lwvn</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lwtt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lwttm</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lwvd</th>
<th>1/2/3/4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2/3/4/5</td>
</tr>
<tr>
<td></td>
<td>14/21/3</td>
</tr>
<tr>
<td></td>
<td>88/3</td>
</tr>
<tr>
<td>lwvn</td>
<td>1/2/3/4</td>
</tr>
<tr>
<td></td>
<td>2/3/1/2</td>
</tr>
<tr>
<td></td>
<td>88/1/7</td>
</tr>
<tr>
<td>lwtt</td>
<td>1/2</td>
</tr>
<tr>
<td></td>
<td>4/52</td>
</tr>
<tr>
<td></td>
<td>14/142</td>
</tr>
<tr>
<td>lwttm</td>
<td>1/2</td>
</tr>
<tr>
<td></td>
<td>4/52</td>
</tr>
<tr>
<td></td>
<td>15/157</td>
</tr>
</tbody>
</table>

| Total 21 | 86 | 832 | 2725 |

1. Total number of subroutines = **21**
2. Total number of outer DO loops = **86**
3. Total number of assignment statements inside DO loops = **832**
4. Total number of variable references in DO loops, excluding array subscripts = **2725**
Generating statistics for phylmd/radlwsw.F

<table>
<thead>
<tr>
<th>lwvd</th>
<th>DO LOOP#</th>
<th>ASSIGN STMTS</th>
<th>VAR REFS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lwvn</th>
<th>DO LOOP#</th>
<th>ASSIGN STMTS</th>
<th>VAR REFS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lwtt</th>
<th>DO LOOP#</th>
<th>ASSIGN STMTS</th>
<th>VAR REFS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lwttm</th>
<th>DO LOOP#</th>
<th>ASSIGN STMTS</th>
<th>VAR REFS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Total 21 | 86 | 832 | 2725 |

```python
1 import xml.etree.ElementTree as et
2 root=et.parse('/home/sourav/LMDZ.3.3/libf/phylmd/radlwsw.F.xml').
3 getroot()
4 subrtotal=0
dntotal,antotal,vaARIABLEtotal=0,0,0
9 snodes=root.findall('SUBROUTINE')
10 for s in snodes:
11     print('|{0:>8s} |{1:>8s} |{2:>13s} |{3:>8s} |'.\
12     format(s.get('name'),'DO LOOP#','ASSIGN STMTS','VAR REFS'))
13     dn=0
14     for d in s.iterfind('BODY/DO'):
15         dn+=1
16         an=0; rhsvars=0
17     for a in d.iterfind('./EXEC_ASSIGN'):
18         an+=1
19         rhsvars=+
20         len(a.findall('RHS//EXPR_VARIABLE'))-
21         len(a.findall('RHS//REF_ARRAY//EXPR_VARIABLE'))
22     print('|{0:8s} |{1:8d} |{2:13d} |{3:8d} |'.\
23     format(' ',dn,an,rhsvars+an))
24     antotal+=an; vaARIABLEtotal+=rhsvars+an
25     dntotal+=dn; subrtotal+=1
26
27 print('-'*46)
28 print('|{4:5s} |{0:2d} |{1:8d} |{2:13d} |{3:8d} |'.\
29     format(subrtotal,dntotal,antotal,vaARIABLEtotal,'Total'))
```
Next challenge: analyzing and transforming loops

- Detection of parallelizable loops
  - Criteria for the “parallelizableness” of a loop?
- Loop fission and fusion
- Kernel generation from loops
What needs to be automated

- The full analysis of dependence between iterations requires the theory developed in the field of “Polyhedral Analysis of Loops”.

- Array subscripts should be an affine/linear function of iterator variables and certain integer parameters like the bounds of a loop.

- Conditional statements within loops allowed under certain conditions.

- Data dependent array accesses not allowed.
Target language and code structure

- **Target language:**
  - Presently, CUDA Fortran
  - Eventually, CUDA C

- **Structure of generated code:**
  - What was manually achieved via Approach 2
  - Preferably, human readable:
    - Emphasis on retaining original variable names
    - Annotation of statements in the generated code about their original location in the model
Conclusion

- The time spent in the laborious manual writing and debugging of CUDA Fortran programs would be much better utilized developing programs in higher level languages that automate the generation of those CUDA Fortran programs.

- XML is very convenient as an intermediate representation of Fortran source and greatly facilitates developing tools for automatic program analyses.

- Polyhedral Loop Analysis is an area that needs to be studied by model developers/translators to yield new insights into automatic model parallelization.
Other parallelization efforts at IIT Delhi

- Multi-GPU implementation of LMDZ dynamics using PMLIB (Partitioned Memory Parallel Programming Interface Library), a library built over OpenMPI that provides facility for data distribution and parallel execution.¹

- OpenACC implementation of the physics of LMDZ5.²

¹,²: Dr. Subodh Kumar, CSE, IIT Delhi
Thanks

sourav.mukherjee.iitkgp@gmail.com