KGen Update & Performance Optimization of a Long-wave Radiation Code

Youngsung Kim

Multicore Workshop 2017
● **KGen Enhancements**
  ○ Quantitative metrics for Kernel representativeness
  ○ Extraction code that contains MPI routines

● **Collaborating for performance optimization with a domain scientist**
  ○ Who not only develops algorithm but also involves coding deeply and willingly
  ○ Using Folding analysis to pinpoint source lines for optimizations and to get directions for optimizations
KGen Enhancement Overview

Production Application

An algorithm
KGen Enhancement Overview

Production Application

An algorithm

Automated Kernel extraction

KGen

Kernel with driver code

An algorithm
KGen Enhancement Overview

KGen

Production Application

An algorithm

Automated Kernel extraction

Kernel with driver code

An algorithm
KGen Enhancement Overview

Production Application

An algorithm

Automated Kernel extraction

Kernel with driver code

An algorithm

Measurements of an execution attribute

Measurement histogram

Measurements of an execution attribute

Measurement histogram
KGen Enhancement Overview

KGen

Production Application
An algorithm

Automated Kernel extraction

Kernel with driver code
An algorithm

Measurements of an execution attribute

KGen Extension

Compare using Descriptive Statistics

Measurement histogram
Types of descriptive statistics

<table>
<thead>
<tr>
<th>type</th>
<th>statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central tendency</td>
<td>mean, median, mode</td>
</tr>
<tr>
<td>Variability</td>
<td>min, max, standard deviation, variance</td>
</tr>
<tr>
<td>Shape</td>
<td>kurtosis, skewness</td>
</tr>
</tbody>
</table>

Calculating a metric for difference

\[
DIFF = \frac{STAT_{KERNEL} - STAT_{APP}}{STAT_{APP}} \times 100(\%) \]
Experiment Configuration

● System
  ○ **Yellowstone**: 2.6GHz Intel Xeon E5-2670, 16 cores per node
  ○ Intel Fortran Compiler 16.0.1

● Applications
  ○ Community Earth System Model (CESM)
    ■ Rapid Radiative Transfer Model for General Circulation Models (RRTMG) Longwave Radiation Physics with 630 MPI ranks.
  ○ Linear Algebra Package (LAPACK)
    ■ Singular Value Decomposition Routine (DGESVD)
    ■ Sequential Test Program
## RRTMG Longwave Kernel

- Elapsed time for RRTMG Longwave kernel

<table>
<thead>
<tr>
<th>Stat. type</th>
<th>Value (a)</th>
<th>Value (b)</th>
<th>Diff(%) (a) ~ (b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean (ms)</td>
<td>6.44</td>
<td>4.12</td>
<td>-36</td>
</tr>
<tr>
<td>(min., max.)</td>
<td>(3.84, 7.8)</td>
<td>(3.83, 4.95)</td>
<td>(-0.3, -36.5)</td>
</tr>
</tbody>
</table>

![Frequency distributions for RRTMG Longwave on CESM and RRTMG Longwave kernel](image.png)
### Elapsed time for RRTMG Longwave kernel

<table>
<thead>
<tr>
<th>Stat. type</th>
<th>Value (a)</th>
<th>Value (b)</th>
<th>Diff(%) (a) ~ (b)</th>
<th>Value (d)</th>
<th>Diff(%) (a) ~ (d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean (ms)</td>
<td>6.44</td>
<td>4.12</td>
<td><strong>-36</strong></td>
<td>6.35</td>
<td><strong>-1.4</strong></td>
</tr>
<tr>
<td>(min., max.)</td>
<td>(3.84 7.8)</td>
<td>(3.83, 4.95)</td>
<td>(-0.3, <strong>-36.5</strong>)</td>
<td>(4.02, 7.9)</td>
<td>(-4.7, -1.3)</td>
</tr>
</tbody>
</table>
● PAPI cache counters for the same SVD routine

- L1 cache misses
- L2 cache misses
- L3 cache misses

- Not fully automated yet.
- User may need to modify build process to link with PAPI library
● **Original code**

```fortran
MODULE calc_mod  
  PUBLIC calc  
CONTAINS  
SUBROUTINE calc(i, j, output, out2, out3)  
  INTEGER, INTENT(IN) :: i, j  
  real, INTENT(OUT), dimension(:,:) :: out3, output, out2  
IF ( i > j ) THEN  
  output(i,j) = i - j  
  out2(i, j) = 2*(i-j)  
  out3(i, j) = 3*(i-j)  
ELSE  
  output(i,j) = j - i  
  out2(i, j) = 2*(j-i)  
  out3(i, j) = 3*(j-i)  
END IF  
END SUBROUTINE  
END MODULE  
```

● **KGen-annotated code**

```fortran
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!  
!! 2 conditional blocks exist in this file  
!! 2 conditional blocks are executed at least once among all  
the conditional blocks.  
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!  

MODULE calc_mod  
  PUBLIC calc  
CONTAINS  
  SUBROUTINE calc(i, j, output, out2, out3)  
    ...  
  IF ( i > j ) THEN  
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!  
   !! Total number of visits: 60  
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!  
    ELSE  
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!  
   !! Total number of visits: 100  
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!  
    END IF  
  END SUBROUTINE  
END MODULE  
```
update_mod.F90(user-annotated)

```fortran
MODULE update_mod
...
CONTAINS
SUBROUTINE update(rank, N)
...
!$KGEN begin_callsite calc
  gsum = 0
  DO i=1, COL
    DO j=1, ROW
      CALL calc(i, j, output)
    END DO
  END DO
  lsum = SUM(output)
!$KGEN exclude
CALL mpi_gather(lsum, ...)
!$KGEN write(gsum)
IF (rank==0) THEN
  print *, 'global sum=', SUM(gsum)
END IF
!$KGEN end_callsite calc
END SUBROUTINE
END MODULE
```

update_mod.F90 (kernel)

```fortran
MODULE update_mod ...
CONTAINS
SUBROUTINE update(kgen_unit)
  INTEGER, INTENT(IN) :: kgen_unit
  ... READ(UNIT=kgen_unit) i 
  ... DO i=1, COL
    ... CALL calc(i, j, output) ! call to kernel
    ... END DO
  lsum = SUM(output)
  CALL kgen_read_integer_4_dim1(gsum, ...)
  IF (rank==0) THEN...
  END IF
  CALL kgen_verify_integer_4_dim2(...)
  CONTAINS
    SUBROUTINE kgen_verify_integer_4_dim2(...)
      ...
      IF (ALL(var==ref_var)) THEN ...
      END IF
    END SUBROUTINE
  END SUBROUTINE
END MODULE
```
Available at
http://github.com/NCAR/KGen

Will be published as
“Assessing Representativeness of Kernels using Descriptive Statistics”, WRAp 2017
RRTMGP-LW
Performance Analysis & Optimization
● Collaboration with a domain scientist
  ○ Who not only develops algorithm but also involves coding deeply and willingly
  ○ A domain scientist updates codes while a performance engineer analyses and suggests optimizations

● RRTMGP-Longwave
  ○ Longwave part of “A High-Performance Broadband Radiation Code for the Next Decade”
  ○ Entirely new code design with new Fortran 2003 features
  ○ Incorporates recent science achievements
  ○ Initially about five times slower than RRTMG
### Code structure

#### RRTMG-LW

DO iplon=1, ncol
  
inatm()
  cldprmc()
  setcoef()
  taumol()
  rtnmc()
  
  DO iband=istart, iend
    1000 continue
    DO lev=nlayer,1,-1
      drad(lev-1) = ...
    if (...) GO TO 1000

#### RRTMGP-LW

gas_optics()
  
  compute_gas_taus()
  source()

lw_solver()
  
  lw_solver_noscat()
   DO igpt=1, ngpt
    DO ilev=nlay,1,-1
      radn_dn(:,ilev,igpt) = ...
  lw_solver() : same as above

Elapsed time: 10.56 ms

Elapsed time: 51.6 ms
Total # of loop iterations for inner-most loops

**RRTMG-LW**

```
DO iplon=1, ncol  = 31
    inatm()
    cldprmc()
    setcoef()
    taumol()
    rtnmc()
    DO iband=istart, iend  = 16
    1000 continue
```

```
DO lev=nlayer,1,-1  = 16
    drad(lev-1) = ...
    if (...) GO TO 1000
```

```
31 * 16 * 8.75 * 16 = 69440
```

**RRTMGP-LW**

```
gas_optics()
    compute_gas_taus()
    source()
    lw_solver()
    lw_solver_noscat()
    DO igpt=1, ngpt  = 256
        DO ilev=nlay,1,-1  = 16
            radn_dn(:,ilev,igpt) = ...
        = 31
    lw_solver() : same as above
```

```
256 * 16 * 31 * 2 = 492032
```

There exists about 7 times more iterations on RRTMGP
- Distinct regions in RRTMGP-LW execution profile
- Folding analysis can match each regions with corresponding source lines
The most expensive code lines in `lw_solver()`

- In “mo_rrtmgp_solver_kernels.F90”

<table>
<thead>
<tr>
<th>Line #</th>
<th>% time of region C or E</th>
<th>Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>84</td>
<td>8.59</td>
<td>$\text{tau}<em>\text{loc}(\cdot,\text{i}</em>{\text{lev}}) = \max(\text{tau}(\cdot,\text{i}<em>{\text{lev}},\text{ig}</em>{\text{pt}})/\mu(\cdot,\text{ig}<em>{\text{pt}}),2.</em>\text{wp}*\text{tiny}(1._\text{wp}))$</td>
</tr>
<tr>
<td>87</td>
<td>26.41</td>
<td>$\text{trans}(\cdot,\cdot) = \exp(-\text{tau}_\text{loc}(\cdot,\cdot))$</td>
</tr>
<tr>
<td>159</td>
<td>13.79</td>
<td>$\text{lay}<em>\text{emission} = \text{merge}(\text{lev}</em>\text{src} + 2.<em>\text{wp} \times (\text{lay}</em>\text{src} - \text{lev}<em>\text{src}) \times (1.</em>\text{wp}/\text{tau} - \text{trans}/(1.<em>\text{wp} - \text{trans})), &amp; 0.</em>\text{wp}, \text{trans} &lt; 1.<em>\text{wp} - \text{spacing}(1.</em>\text{wp}))$</td>
</tr>
<tr>
<td>160</td>
<td>15.00</td>
<td></td>
</tr>
<tr>
<td>161</td>
<td>20.85</td>
<td></td>
</tr>
</tbody>
</table>
The most expensive code lines in `compute_gas_tau_core()` and `source()`

- In “mo_gas_optics_kernels.F90”

<table>
<thead>
<tr>
<th>Line #</th>
<th>% time of region A</th>
<th>Codes</th>
</tr>
</thead>
</table>
| 469 ~ 483 | 59.5 | `res = &
scaling(1) * &
( &
fmajor(1,1,1) * k(jeta(1), jpress-1, jtemp ) + &
fmajor(2,1,1) * k(jeta(1)+1, jpress-1, jtemp ) + &
fmajor(1,2,1) * k(jeta(1), jpress , jtemp ) + &
fmajor(2,2,1) * k(jeta(1)+1, jpress , jtemp ) &
) + &
scaling(2) * &
( &
fmajor(1,1,2) * k(jeta(2), jpress-1, jtemp+1) + &
fmajor(2,1,2) * k(jeta(2)+1, jpress-1, jtemp+1) + &
fmajor(1,2,2) * k(jeta(2), jpress , jtemp+1) + &
fmajor(2,2,2) * k(jeta(2)+1, jpress , jtemp+1) &
)` |
Cache characteristics

- High L1 and L2 cache misses are resolved by L3 cache
- Folding analysis tells that the high cache miss regions are caused by mostly two assignment statements in gas optics.
Both of RRTMG and RRTMGP do not utilize AVX instruction
With “-xAVX” compiler flag, around 8% of speedup is achieved on both of them
● In the region of RRTMGP “solvers”, around 3% of total instructions are floating-point division instructions
● In addition to obvious division operations, EXP() intrinsic function is another main cause
During a meeting with domain scientists, the analysis outcomes and several suggestions for optimization were shared
  - The amount of computation is about 4.4 times higher on RRTMGP
  - Source codes that caused high L1 data cache miss were identified with explanations of possible causes
  - Recommendations were made for reducing floating-point operations and increasing vectorization
First optimization done by a domain scientist

- Optimized code patch was handed over to a performance engineer for analysis
- # of floating division instructions are reduced

- Compact usage of “SUM” built-in function seems not helpful

- A part of code runs faster, but the other part slower
The issue of high L2 Cache misses was still remained
  - SUBROUTINE “source” is major source of the high L2 Cache write region
  - FUNCTION “reorder_wp_123x321” is also contributing to the high L2 cache misses
Suggestions to reduce cache misses

- A performance engineer applied optimizations of “precalculation” and “loop interchange” to his code base.

- It is simulated that around 10% speedup is possible with the optimizations
- Modified code by a performance engineer was sent to a domain scientist
Second optimization (ver6) done by a domain scientist

- Significant performance enhancements were achieved in the area of vectorization and floating-point division (FDV)

- Ver3: original RRTMGP LW version
- Ver6: Done by a domain scientist
- Ver7: Ver6 + Suggestions from previous analysis on gas-optics

- Modified code by a performance engineer again was sent to a domain scientist
Third optimization (ver8) done by a domain scientist

- A domain scientist sent his “finalish” version to a performance engineer for analysis
- Vectorization enhancement were disappeared!

- Ver8 shows similar vectorization to Ver3 (original version)
Third optimization (ver8) done by a domain scientist

- Improvement on floating-point division was also vanished!
What happened was:

- Code base that the domain scientist used was different from what a performance engineer worked on
  - The original code based are managed within AER version control system
  - The performance engineer used a different code base which was originally the same code based with one at AER but diverged afterward
- Patch file from a domain scientist did not include all source files that are diverged from original code base
- As a result, previous optimizations were discarded as the patch files only contains the newest optimizations
● Optimization summary
  ○ Before starting optimization, check if the amount of computation is reasonable
  ○ Folding analysis combined with PAPI can pinpoint code lines to be optimized as well as directions for optimizations
  ○ EXP() intrinsic function and division operation are expensive

● Collaboration with a domain scientist
  ○ Domain scientist can change algorithm itself which can impact the performance significantly
  ○ Better to have one common code repository
  ○ It may not be helpful to share all the details of technical analysis with a domain scientist
Thank you!

Q & A
Back-up slides
Content of measurements

- Invocation Triplets
  - MPI rank number
  - OpenMP thread ID
  - Invocation order

- Runtime attributes
  - Elapsed time
  - PAPI counter
  - Source code coverage
  - Will add more attributes in future

Saved as 4-ary tuples of

( “MPI rank”, “OpenMP TID”, “Invocation order”, “runtime attribute”)
Elapsed time is measured for SVD routines of LAPACK

- Increasing workload from 64X64 matrix to 120X92

<table>
<thead>
<tr>
<th>Stat. type</th>
<th>SVD</th>
<th>SVD kernel</th>
<th>Diff(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean (ms)</td>
<td>114</td>
<td>118</td>
<td>2.9</td>
</tr>
<tr>
<td>(min., max.)</td>
<td>(1.0, 498)</td>
<td>(2.7, 539)</td>
<td>(-17.1, 8.3)</td>
</tr>
<tr>
<td>Std. dev.</td>
<td>0.46</td>
<td>0.46</td>
<td>0.1</td>
</tr>
<tr>
<td>Skew</td>
<td>1.07</td>
<td>0.98</td>
<td>-8.5</td>
</tr>
</tbody>
</table>
**KGen Overview**

### Usual Tasks
- Configuration
- Input data preparation
- Building dependencies
- Compilation/linking
- Submitting to queue
- Post-processing

### KGen does
- Parsing Fortran Code
- Analyzing on ASTs
- Extracting code
- Instrumenting for data

### ASTs: Abstract Syntax Trees

### KGen Kernels
- Stand-alone Executable
- In/out data from app.
- Verification
- Timing
- Perturbation
- Makefile for build/run
KGen Extension Concept

**Fortran Application**

- A code block
  - Subroutine
  - Function
  - Do loop
  - Other code block

**Measurement distribution**

- Begin measurement
  - Elapsed time
  - PAPI counters
  - ETC.

- Stop measurement

**Automated Kernel extraction**

**KGen**

**KGen kernel**

- Executable code
- Preserve code structure

**In/out state**

↓: Program Execution  ☐: Data Collection
**KGen Extension Concept**

**Fortran Application**
- A code block
  - Subroutine
  - Function
  - Do loop
  - Other code block

**KGen kernel**
- Executable code
- Preserve code structure

**Measurement distribution**
- Begin measurement
  - Elapsed time
  - PAPI counters
  - ETC.

**Stop measurement**

Use the distribution to correctly generate input/output data to ensure representation.

![Diagram of program execution and data collection](image-url)
KGen Extension Concept

**Fortran Application**

- A code block
  - Subroutine
  - Function
  - Do loop
  - Other code block

**KGen**

- Measurement distribution
- Automated Kernel extraction

**KGen kernel**

- Executable code
- Preserve code structure

**Measurement distribution**

- Begin measurement
  - Elapsed time
  - PAPI counters
  - ETC.

- Stop measurement

**Verify similarity of two distributions**

- average
- min/max
- shape

**Use the distribution to correctly generate input/output data to ensure representation.**

**Program Execution**

[\(\downarrow\) : Program Execution]

**Data Collection**

[\(\uparrow\) : Data Collection]
**MG2 kernel**

- **Elapsed time for MG2 kernel**
  - MG2 kernel is one of time consuming cloud physics routine in CESM*

<table>
<thead>
<tr>
<th>Stat. type</th>
<th>Value (a)</th>
<th>Value (b)</th>
<th>Diff(%) (a) ~ (b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean (ms)</td>
<td>1.67</td>
<td>0.75</td>
<td>-55</td>
</tr>
<tr>
<td>(min., max.)</td>
<td>(1.0, 2.5)</td>
<td>(0.6, 1.0)</td>
<td>(-42.5, -62.4)</td>
</tr>
<tr>
<td>Std. dev.</td>
<td>0.12</td>
<td>0.47</td>
<td>299.7</td>
</tr>
<tr>
<td>Skew</td>
<td>-0.76</td>
<td>-0.44</td>
<td>-42.2</td>
</tr>
</tbody>
</table>
Elapsed time for MG2 kernel

- MG2 kernel is one of time consuming cloud physics routine in CESM

<table>
<thead>
<tr>
<th>Stat. type</th>
<th>Value (a)</th>
<th>Value (b)</th>
<th>Diff(%) (a) ~ (b)</th>
<th>Value (d)</th>
<th>Diff(%) (a) ~ (d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean (ms)</td>
<td>1.67</td>
<td>0.75</td>
<td>-55</td>
<td>1.16</td>
<td>-30.8</td>
</tr>
<tr>
<td>(min., max.)</td>
<td>(1.0, 2.5)</td>
<td>(0.6, 1.0)</td>
<td>(-42.5, -62.4)</td>
<td>(0.65, 1.53)</td>
<td>(-34.6, -38.8)</td>
</tr>
<tr>
<td>Std. dev.</td>
<td>0.12</td>
<td>0.47</td>
<td>299.7</td>
<td>0.35</td>
<td>198</td>
</tr>
<tr>
<td>Skew</td>
<td>-0.76</td>
<td>-0.44</td>
<td>-42.2</td>
<td>-0.54</td>
<td>-29</td>
</tr>
</tbody>
</table>
Analysis of the first optimization and further suggestions

- First optimization has both of positive and negative impact on performance
- With applying optimizations to “source” subroutine and reorder123x321 function, around 9.8% speedup would be achieved compared to RRTMPG LW V3
- It is desirable to “remove” reordering of major arrays as many as possible
- In several places, array indices are not arranged for unit-stride
  - EX: $\text{lay}_{\text{src}}(icol,i\text{lay},\text{band2gpt}(1,ibnd):\text{band2gpt}(2,ibnd)) = \&$
  - $\text{pfrac}($\text{band2gpt}(1,ibnd):\text{band2gpt}(2,ibnd),icol,i\text{lay}) \times \text{planck}_\text{function}(ibnd)$
Analysis of the second optimization and further suggestions

- The second update shows significant performance improvement on LW solver part
  - ~18% of overall speed-up from 24.8 ms (ver3) to 20.38 ms (ver6)

- Good ...

- On top of Robert’s update, previous update on “gas optics” is merged
  - Additional ~14% of speed-up from 20.38 ms (ver6) to 17.5 ms (ver7)

- Altogether, after applying both of updates, the final version shows ~30% speed-up
  - /glade/p/tdd/asap/kgen_kernels/port/rrtmgp14_cam5_4_48/rrtmgp_lw.v7
● In “mo_gas_optics_kernels.F90”

<table>
<thead>
<tr>
<th>Line #</th>
<th>% time of region B</th>
<th>Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>359</td>
<td>52.45</td>
<td>lev_src_inc(icol,ilay,band2gpt(1,ibnd):band2gpt(2,ibnd)) = &amp;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pfrac(band2gpt(1,ibnd):band2gpt(2,ibnd),icol,ilay-1) * planck_function(ibnd)</td>
</tr>
<tr>
<td>361</td>
<td>42.88</td>
<td>lev_src_dec(icol,ilay,band2gpt(1,ibnd):band2gpt(2,ibnd)) = &amp;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pfrac(band2gpt(1,ibnd):band2gpt(2,ibnd),icol,ilay) * planck_function(ibnd)</td>
</tr>
</tbody>
</table>

● In “mo_fluxes_byband.F90A” and “mo_optical_props_kernels.F90B”

<table>
<thead>
<tr>
<th>Line #</th>
<th>% time of region D</th>
<th>Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>123A</td>
<td>19.31</td>
<td>this%bnd_flux_up(;;,bnd) = sum(gpt Flux up(;;,gpt_lim(1):gpt_lim(2)), dim=3)</td>
</tr>
<tr>
<td>132A</td>
<td>14.92</td>
<td>this%bnd_flux_dn(;;,bnd) = sum(gpt Flux_dn(;;,gpt_lim(1):gpt_lim(2)), dim=3)</td>
</tr>
<tr>
<td>369B</td>
<td>21.89</td>
<td>tau1(;;;) = tau1(;;;) + tau2(;;;)</td>
</tr>
</tbody>
</table>
Similar L2 Cache behavior on “gas optics” part

L2 Cache accesses for writing are reduced

Instead, L2 cache accesses for reading are increased

Overall, total L2 cache misses are reduced
Vectorization

- Both of them do not utilize AVX instruction
- Reran them with following compiler flags
  - -fp-model fast
  - -O3
  - -xAVX
- Speed-ups
  - RRTMG: 10.56 ms -> 9.36 ms
  - RRTMGP: 51.6 ms -> 46.66 ms
Other performance characteristics

- Around 3% of total instructions at lw_solver() of RRTMGP are floating-point division instructions.
- After converting divisions to multiplications, around 3.5% of speedup is achieved.

After converting divisions to multiplications related to mu/mus at line # 84 of “mo_rrtmgp_solver_kernels.f90”
A suggestion by a performance engineer

- "Original"

! compute level Planck source function for each g-point in increasing ilay direction
   do icol = 1, ncol
     do ilay = 2, nlay
       planck_function(:) = interpolate1D(tlev(icol,ilay), temp_ref_min, totplnk_delta, totplnk)
       do ibnd = 1, nbnd
         lev_src_inc(icol,ilay,band2gpt(1,ibnd):band2gpt(2,ibnd)) = &
         pfrac(band2gpt(1,ibnd):band2gpt(2,ibnd),icol,ilay-1) * planck_function(ibnd)
       end do
     end do ! ilay
   end do ! icol

- Optimized

! precalculate planck_function
   do ilay = 1, nlay + 1
     do icol = 1, ncol
       planck_function(:) = interpolate1D(tlev(icol,ilay), temp_ref_min, totplnk_delta, totplnk)
       do ibnd = 1, nbnd
         planck_function2(icol, ilay, ibnd) = planck_function(ibnd)
       end do ! ibnd
     end do ! icol
   end do ! ilay

! loop interchange, and loop merge for edge cases
   do ibnd = 1, nbnd
     do igpt = band2gpt(1,ibnd), band2gpt(2,ibnd)
       do ilay = 1, nlay + 1
         do icol = 1, ncol
           lev_src_inc(icol,ilay,igpt) = pfrac(igpt,icol,ilay-1) * planck_function2(icol, ilay, ibnd)
           lev_src_dec(icol,ilay,igpt) = pfrac(igpt,icol,ilay) * planck_function2(icol, ilay, ibnd)
         end do ! icol
       end do ! ilay
     end do ! igpt
   end do ! ibnd

lev_src_inc(:,1,:) = 0 ! this value is padding
lev_src_dec(:,nlay+1,:) = 0 ! this value is padding
Another suggestion by a perf. engineer

- “Original”

```fortran
! reorder the indecies of 4D array
function reorder_wp_123x321(array)
  real(wp), dimension(:,:,,:), intent(in) :: array
  real(wp), dimension(size(array,dim=3),size(array,dim=2),size(array,dim=1)) :: reorder_wp_123x321
  integer :: i1
  do i1 = 1, size(array,dim=1)
    reorder_wp_123x321(:,:,i1) = transpose(array(i1,:,:))
  end do
end function
```

- Optimized

```fortran
! used literal number for loop iterations
function reorder_wp_123x321(array)
  real(wp), dimension(:,:,,:), intent(in) :: array
  real(wp), dimension(16,31,256) :: reorder_wp_123x321
  integer :: i1, i2, i3
  do i1 = 1, 256
    do i3 = 1, 16
      do i2 = 1, 31
        reorder_wp_123x321(i3,i2,i1) = array(i1,i2,i3)
      end do
    end do
  end do
end function
```