Investigating and Vectorizing IFS on a Cray Supercomputer

Ilias Katsardis (Cray)
Deborah Salmond, Sami Saarinen (ECMWF)

17th Workshop on High Performance Computing in Meteorology
24-28 October 2016

Introduction

- **About me**

Ilias Katsardis

Application Analyst in CRAY

- **The task**

Support for ECMWF

Currently: Vectorize IFS for BDW and Investigate the code

- **Hardware info**

ECMWF node: 36 cores @ 2.1GHz with 45M cache

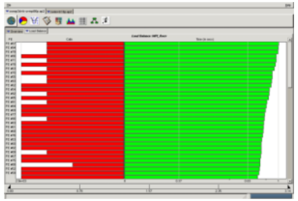
KNL node: 68 cores @ 1.4GHz

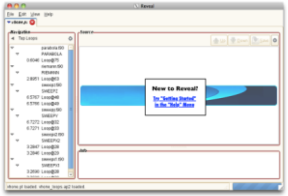
Contents

- Part I: FLOP counting
- Part II: Optimizations on IFS
- Part III: Same optimizations on KNL

Allies on this task

- **Cray Source Listing** (*.lst files)
- **Cray PAT**
- **Cray Apprentice2** - A graphical analysis tool that can be used to visualize and explore the performance data captured during program execution.
- **Reveal** - A graphical source code analysis tool that can be used to correlate performance analysis data with annotated source code listings, to identify key opportunities for optimization.
- **Dr. Hook**
- Flop counting either with **CrayPAT** (PAPI) or **Intel SDE**
- **Intel Tools**





FLOP counting Intro

- First we need to verify the accuracy of FLOP counting with a known number of FLOPs before we try and count IFS FLOPs.
- For that we opted in for an HPL run instead of a known number of loops, data, etc that would allow us to calculate the number of Operations.
- The reason for that we needed to be sure that our procedure works with a number of other libraries and optimizations (e.g. –O3 and MKL).

FLOP counting with Intel SDE (how to)

1. **We need a wrapper like this:**

```
#!/bin/bash
RANK=$ALPS_APP_PE
if [ $RANK == 1  ]
then
./sde64 -d -bdw -iform 1 -omix
myapp_bdw_mix.out -top_blocks
30  -- $rundir/xhplfinal
else
$rundir/xhplfinal
fi
```

2. **We need to have a job script with:**

```
export PMI_NO_FORK=1
export
LD_LIBRARY_PATH=sdelib_path:$LD_LI
BRARY_PATH
aprun -b -cc none -j 1 -n 72 -N 36
-d $OMP_NUM_THREADS ./wrapper.sh
```

FLOP counting with Intel SDE (Results)

- Launching like that will produce an output file with all the counters it could find for example:

```
*elements_fp_double_1                    1271844954
*elements_fp_double_2                    32615811
*elements_fp_double_4                    1086273739672
VFMADD213PD_YMMqq_YMMqq_MEMqq            22162176
VFMADD213SD_XMMdq_XMMq_MEMq              252522001
VFMADD213SD_XMMdq_XMMq_XMMq              505051973
VFMADD231PD_YMMqq_YMMqq_MEMqq            31564800
VFMADD231PD_YMMqq_YMMqq_YMMqq            1074009497376
VFNMADD213PD_YMMqq_YMMqq_YMMqq           2906251600
VFNMADD231PD_YMMqq_YMMqq_YMMqq           2906224800
RESULT in 8643442506646  FLOP or 8643GFLOP (we need to multiply each register with the
correct amount of Operation recquired for each register...)
```

- We are interested for:

```
*elements_fp_(double|singe)_(1|2|4 etc)
VADD(S|P)_(X|Y)MM(d|q)q_etc...
re.search(r"(^VF[N|M])(ADD|SUB)(\d\d\d)(PD|SD|SS|PS)_(X|Y|Z)", line)
```

FLOP counting with Intel SDE (results on HPL)

- Run without SDE reports: **2316 GFLOPS**
- Run with SDE reports: **2297 GFLOPS** $\frac{8643_{FLOP} * 88_{mpitasks}}{331.6_{totalruntime}} = 2297.83_{GFLOPS}$
- Results are within **1%!**
- Take note that with SDE we have to count **initialization** and **finalization** period so actually the results are even more accurate that the 1%.
- Slowdown because of emulating = x2-x4 times

FLOP counting with CrayPAT (how to)

- module load perftools-base perftools
- pat_build binary_name

FLOP counting with CrayPAT (results on IFS)

```
Table 3:  Program HW Performance Counter Data

==========================================================  Total
-----------------------------------------------------------------------------
  CPU_CLK_THREAD_UNHALTED:THREAD_P          1,148,950,914,774
  CPU_CLK_THREAD_UNHALTED:REF_XCLK             49,610,954,365
  DTLB_LOAD_MISSES:MISS_CAUSES_A_WALK             481,754,204
  DTLB_STORE_MISSES:MISS_CAUSES_A_WALK             53,646,594
  L1D:REPLACEMENT                              71,972,039,189
  L2_RQSTS:ALL_DEMAND_DATA_RD                  13,713,375,918
  L2_RQSTS:DEMAND_DATA_RD_HIT                   7,179,862,200
  MEM_UOPS_RETIRED:ALL_LOADS                  787,829,823,118
  FP_ARITH:SCALAR_DOUBLE                2,512,276,465
  FP_ARITH:128B_PACKED_DOUBLE             432,030,907
  FP_ARITH:256B_PACKED_DOUBLE   2,710,780,379,690
  CPU_CLK                              2.32GHz
  HW FP Ops / User time       21,422.429M/sec  10,846,497,857,039 ops
  Total DP ops                      21,422.429M/sec
10,846,497,857,039 ops
  Computational intensity                    ops/cycle          13.77 ops/ref
  MFLOPS (aggregate)   21,422.43M/sec
  TLB utilization               1,471.48 refs/miss          2.87 avg uses
  D1 cache hit,miss ratios         90.9% hits              9.1% misses
  D1 cache utilization (misses)    10.95 refs/miss          1.37 avg hits
  D2 cache hit,miss ratio          90.9% hits              9.1% misses
  D1+D2 cache hit,miss ratio       99.2% hits              0.8% misses
  D1+D2 cache utilization         120.58 refs/miss         15.07 avg hits
  D2 to D1 bandwidth           1,653.117MiB/sec   877,656,058,752 bytes
```

21,422.43 (M/sec per task) x 72 (task)= 1.542e+e03

Or else…. 99.74% accurate based on HPL output!

```
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WC01C2C4      104832   192     4    18             496.87             1.546e+03
HPL_pdgesv() start time Sun Oct 16 13:47:35 2016

HPL_pdgesv() end time   Sun Oct 16 13:55:52 2016

--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV-
Max aggregated wall time rfact . . . :               3.01
+ Max aggregated wall time pfact . . :               2.27
+ Max aggregated wall time mxswp . . :               2.13
Max aggregated wall time update  . . :             474.54
+ Max aggregated wall time laswp . . :              18.78
Max aggregated wall time up tr sv  . :               0.26
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0018303 ...... PASSED
================================================================================
```

FLOPs with IFS and SDE (long runs)

- Output from TCo1279 L137 (2816 MPI x 9 Threads - hyper-threading on) or 352 Nodes
- Total GFLOP counted = 11694.94
- Runtime input in seconds was : 2351
- Approximated (per task) GFLOPS : 4.974
- Approximated TOTAL GFLOPS: 14008 or 3.29% of peak performance
- TOTAL Bytes read: 374 Tb
- TOTAL Bytes wrote: 26.7 Tb

IFS profile

- Broadwell 36 core original run gives following TOP 10:

MPI-tasks : 6 | OpenMP-threads : 6

Wall-times over all MPI-tasks (secs) : Avg=231.552, StDev=0.326

Avg-%	Avg.time	Min.time	Max.time	St.dev	Imbal-%	# of calls : Name of the routine
8.96%	20.748	20.304	21.246	0.315	4.43%	805680 : CLOUDSC
4.49%	10.395	9.717	11.031	0.481	11.91%	402840 : CPG
3.94%	9.111	7.876	10.058	0.961	21.69%	2014200 : LASCAW
3.73%	8.628	8.172	9.038	0.328	9.58%	367992173 : CUADJTQ
2.84%	6.581	6.202	7.435	0.545	16.58%	10071000 : LAITLI
2.59%	5.994	5.796	6.287	0.217	7.81%	2014200 : LARCHE
2.55%	5.894	5.267	6.289	0.394	16.25%	402840 : CLOUDVAR
2.39%	5.526	5.491	5.552	0.024	1.10%	402840 : CALLPAR
2.28%	5.282	4.784	6.328	0.721	24.40%	2014200 : LAITRI
2.16%	5.007	2.209	8.308	2.700	73.41%	13140 : >OMP-TRLTOGPACK(1605)

TOP 10 is 35.93% of the whole run. TOP 25 is ~56%...

The optimization task

You want a flag or trick like this

The optimization task

You want a flag or trick like this

But reality is more like this

BDW-CLOUDSC investigation

- Code like this:
- ZTMPA = 1.0_JPRB/ MAX(ZA(JL,JK),ZEPSEC)
- Causes Floating Point trapping exceptions preventing the compiler to vectorize it.
- For CLOUDSC we have 29 total compiler msgs like this:
- A loop starting at line XXX was not vectorized because of a potential hazard in conditional code on line XXX
- Solution? Add –hnofp_trap add –K trap=none
- We also added !DIR$ LOOP_INFO EST_TRIPS(16) since we know our do loops are 16 due to the NPROM setting.

BDW-CLOUDSC results

Before Optimization:

Wall-times over all MPI-tasks (secs) : Avg=231.552, StDev=0.326

Avg-%	Avg.time	Min.time	Max.time	St.dev	Imbal-%	# of calls : Name of the routine
8.96%	20.748	20.304	21.246	0.315	4.43%	805680 : CLOUDSC

After Optimization:

Wall-times over all MPI-tasks (secs) : Avg=228.972, StDev=0.722

Avg-%	Avg.time	Min.time	Max.time	St.dev	Imbal-%	# of calls : Name of the routine
7.90%	18.085	17.805	18.405	0.224	3.26%	805680 : CLOUDSC

We now have 0 non vectorized functions due to potential hazards.

CLOUDSC is now 13% faster.

We do 2.58s less in total runtime.

BDW-CPG investigation

- **Most time consuming loop in CPG:**

```
            IF (LDUSEPB1) THEN
 + 1-------<   DO JFLD=1,NFLDSLB1
   1           !DIR$ PREFERVECTOR
   1           !DIR$ IVDEP
   1           !DIR$ NOINTERCHANGE
 + 1 Vpr2--<     DO JROF=KSTGLO,MIN(KSTGLO-1+NPROMA,KGPCOMP)
   1 Vpr2          PB1(YDSL%NSLCORE(JROF),JFLD)=ZSLBUF1AU(JROF-KSTGLO+1,JFLD)
   1 Vpr2-->     ENDDO
   1------->   ENDDO
            ENDIF
```

BDW-CPG investigation

- **Vectorize by the outer loop:**

```
            IF (LDUSEPB1) THEN
Vb------<   DO JFLD=1,NFLDSLB1
Vb            !DIR$ PREFERVECTOR !DIR$ NEXTSCALAR
Vb            !DIR$ IVDEP
Vb            !DIR$ NOINTERCHANGE
Vb br4--<     DO JROF=KSTGLO,MIN(KSTGLO-1+NPROMA,KGPCOMP)
Vb br4          PB1(YDSL%NSLCORE(JROF),JFLD)=ZSLBUF1AU(JROF-KSTGLO+1,JFLD)
Vb br4-->     ENDDO
Vb------>   ENDDO
            ENDIF
```

BDW-CPG results

Before Optimization:

Wall-times over all MPI-tasks (secs) : Avg=231.552, StDev=0.326

Avg-%	Avg.time	Min.time	Max.time	St.dev	Imbal-%	# of calls : Name of the routine
4.49%	10.395	9.717	11.031	0.481	11.91%	402840 : CPG

After Optimization:

Wall-times over all MPI-tasks (secs) : Avg=229.057, StDev=0.460

Routines whose total time (i.e. sum) > 1.000 secs will be included in the listing

Avg-%	Avg.time	Min.time	Max.time	St.dev	Imbal-%	# of calls : Name of the routine
2.70%	6.178	5.866	6.386	0.244	8.14%	402840 : CPG

CPG now moves from 2nd most time consuming to 5th

CPG is now 41% faster.

Saving us more than 3s in total runtime.

BDW-LASCAW investigation

```
 DO JLEV=KSLEV,KFLEV+KSLEV-1
   IF (KHOR == 0) ILEV=JLEV
   IF (KHOR == 1) ILEV=KFLDN
    ! * Calculation of linear weights, KL0, KLH0.
!CDIR NODEP
!DIR$ PREFERVECTOR
   DO JROF=KST,KPROF
     IJ_ = MOD(JROF+1-KST,JPDUP)+1
     IZLAT =INT(P4JP*(PIS2-PLAT(JROF,JLEV))+0.75_JPRB)-YDSL%NFRSTLOFF
    ILA   =IZLAT+NINT(SIGN(0.5_JPRB,ZRLATI_(IJ_,IZLAT)-PLAT(JROF,JLEV))-1.5_JPRB)
     ! meridional interpolation: linear weights and input for cubic weights (LASCAW_CLA)
    ! general case
    PDLAT(JROF,JLEV)=(PLAT(JROF,JLEV)-ZRLATI_(IJ_,ILA+1))&
     & /(ZRLATI_(IJ_,ILA+2)-ZRLATI_(IJ_,ILA+1))
    ZDA   =PLAT(JROF,JLEV)-ZRLATI_(IJ_,ILA)
    ZDB   =PLAT(JROF,JLEV)-ZRLATI_(IJ_,ILA+1)
    ZDC   =PLAT(JROF,JLEV)-ZRLATI_(IJ_,ILA+2)
    ZDD   =PLAT(JROF,JLEV)-ZRLATI_(IJ_,ILA+3)
    IILA(JROF,JLEV)=ILA
    ZZWH(JROF,1,JLEV)=(ZDA*ZDC)*ZDD*ZRIPI0_(IJ_,ILA+1)
    ZZWH(JROF,2,JLEV)=(ZDA*ZDB)*ZDD*ZRIPI1_(IJ_,ILA+1)
    ZZWH(JROF,3,JLEV)=(ZDA*ZDB)*ZDC*ZRIPI2_(IJ_,ILA+1)
     ! COMAD meridional interpolation
    IF (LLCOMADH) THEN
      PDLAMAD(JROF,JLEV)=PDLAT(JROF,JLEV)*ZSTDDISV(JROF,JLEV) + 0.5_JPRB *
(1._JPRB-ZSTDDISV(JROF,JLEV))
      ZDA = (ZDA-ZDB)/ZSTDDISV(JROF,JLEV)
      ZDD = (ZDD-ZDC)/ZSTDDISV(JROF,JLEV)
      ZDB = ZDB +0.5_JPRB*(ZRLATI_(IJ_,ILA+2)-ZRLATI_(IJ_,ILA+1)) * &
       & (1._JPRB-ZSTDDISV(JROF,JLEV)) /ZSTDDISV(JROF,JLEV)
      ZDC = ZDC -0.5_JPRB*(ZRLATI_(IJ_,ILA+2)-ZRLATI_(IJ_,ILA+1)) * &
       & (1._JPRB-ZSTDDISV(JROF,JLEV)) /ZSTDDISV(JROF,JLEV)
      ZDA = ZDA + ZDB
      ZDD = ZDD + ZDC
       IILA(JROF,JLEV)=ILA
      ZZWHMAD(JROF,1,JLEV)=(ZDA*ZDC)*ZDD*ZRIPI0_(IJ_,ILA+1)&
       & *ZSTDDISV(JROF,JLEV)*ZSTDDISV(JROF,JLEV)*ZSTDDISV(JROF,JLEV)
      ZZWHMAD(JROF,2,JLEV)=(ZDA*ZDB)*ZDD*ZRIPI1_(IJ_,ILA+1)&
       & *ZSTDDISV(JROF,JLEV)*ZSTDDISV(JROF,JLEV)*ZSTDDISV(JROF,JLEV)
      ZZWHMAD(JROF,3,JLEV)=(ZDA*ZDB)*ZDC*ZRIPI2_(IJ_,ILA+1)&
       & *ZSTDDISV(JROF,JLEV)*ZSTDDISV(JROF,JLEV)*ZSTDDISV(JROF,JLEV)
    ELSE
      PDLAMAD(JROF,JLEV)=PDLAT(JROF,JLEV)
      ZZWHMAD(JROF,1,JLEV)=ZZWH(JROF,1,JLEV)
      ZZWHMAD(JROF,2,JLEV)=ZZWH(JROF,2,JLEV)
      ZZWHMAD(JROF,3,JLEV)=ZZWH(JROF,3,JLEV)
    ENDIF
     ! zonal interpolation: linear weights for 4 lat. cicles
    ! as the grid is regular in the zonal direction,
    ! the cubic weight computation does not need
    ! other input than linear weights (LASCAW_CLO)
    ! general case
    ZLO   =PLON(JROF,JLEV)*ZLSDEPI_(IJ_,ILA  )
    ILO(JROF)   =INT(ZLO )
    PDLO(JROF,JLEV,0)=ZLO -REAL(ILO(JROF) ,JPRB)
    ZLO1  =PLON(JROF,JLEV)*ZLSDEPI_(IJ_,ILA+1)
    ILO1(JROF)  =INT(ZLO1)
    PDLO(JROF,JLEV,1)=ZLO1-REAL(ILO1(JROF),JPRB)
    ZLO2  =PLON(JROF,JLEV)*ZLSDEPI_(IJ_,ILA+2)
    ILO2(JROF)  =INT(ZLO2)
    PDLO(JROF,JLEV,2)=ZLO2-REAL(ILO2(JROF),JPRB)
    ZLO3  =PLON(JROF,JLEV)*ZLSDEPI_(IJ_,ILA+3)
    ILO3(JROF)  =INT(ZLO3)
    PDLO(JROF,JLEV,3)=ZLO3-REAL(ILO3(JROF),JPRB)
     ! COMAD zonal interpolation
    IF (LLCOMADH) THEN
      PDLOMAD(JROF,JLEV,0)=PDLO(JROF,JLEV,0)*ZSTDDISU(JROF,JLEV) + 0.5_JPRB
* (1._JPRB-ZSTDDISU(JROF,JLEV))
      PDLOMAD(JROF,JLEV,1)=PDLO(JROF,JLEV,1)*ZSTDDISU(JROF,JLEV) + 0.5_JPRB
* (1._JPRB-ZSTDDISU(JROF,JLEV))
      PDLOMAD(JROF,JLEV,2)=PDLO(JROF,JLEV,2)*ZSTDDISU(JROF,JLEV) + 0.5_JPRB
* (1._JPRB-ZSTDDISU(JROF,JLEV))
      PDLOMAD(JROF,JLEV,3)=PDLO(JROF,JLEV,3)*ZSTDDISU(JROF,JLEV) + 0.5_JPRB
* (1._JPRB-ZSTDDISU(JROF,JLEV))
    ELSE
      PDLOMAD(JROF,JLEV,0)=PDLO(JROF,JLEV,0)
      PDLOMAD(JROF,JLEV,1)=PDLO(JROF,JLEV,1)
      PDLOMAD(JROF,JLEV,2)=PDLO(JROF,JLEV,2)
      PDLOMAD(JROF,JLEV,3)=PDLO(JROF,JLEV,3)
    ENDIF
     ! vertical interpolation: linear weights
    ! the cubic weight computation are done in
    ! LASCAW_VINTW (including terms for grid irregularity)
    ILEVV=KVAUT(INT(PLEV(JROF,JLEV)*ZFAC))-1
    IF(ILEVV < IFLVM2.AND.&
     & (PLEV(JROF,JLEV)-ZVETA(ILEVV+2)) > 0.0_JPRB) ILEVV=ILEVV+1
    KLEV(JROF,JLEV)=ILEVV
    ! general case
    PDVER(JROF,JLEV)=(PLEV(JROF,JLEV)-ZVETA(ILEVV+1))/&
     & (ZVETA(ILEVV+2)-ZVETA(ILEVV+1))
    ! COMAD vertical interpolation
    IF (LLCOMADV) THEN
      PDVERMAD(JROF,JLEV)=PDVER(JROF,JLEV)*ZSTDDISW(JROF,JLEV) + 0.5_JPRB
* (1._JPRB-ZSTDDISW(JROF,JLEV))
    ELSE
      PDVERMAD(JROF,JLEV)=PDVER(JROF,JLEV)
    ENDIF
     ILO(JROF)=IADDR_(IJ_,ILA  )+YDSL%NSLEXT(ILO(JROF) ,ILA  )
     ILO1(JROF)=IADDR_(IJ_,ILA+1)+YDSL%NSLEXT(ILO1(JROF),ILA+1)
     ILO2(JROF)=IADDR_(IJ_,ILA+2)+YDSL%NSLEXT(ILO2(JROF),ILA+2)
     ILO3(JROF)=IADDR_(IJ_,ILA+3)+YDSL%NSLEXT(ILO3(JROF),ILA+3)
      KL0(JROF,JLEV,0)=ILO(JROF)+YDSL%NASLB1*ILEVV
     KL0(JROF,JLEV,1)=ILO1(JROF)+YDSL%NASLB1*ILEVV
     KL0(JROF,JLEV,2)=ILO2(JROF)+YDSL%NASLB1*ILEVV
     KL0(JROF,JLEV,3)=ILO3(JROF)+YDSL%NASLB1*ILEVV
      KLH0(JROF,JLEV,0)=ILO(JROF)+YDSL%NASLB1*ILEV
     KLH0(JROF,JLEV,1)=ILO1(JROF)+YDSL%NASLB1*ILEV
     KLH0(JROF,JLEV,2)=ILO2(JROF)+YDSL%NASLB1*ILEV
     KLH0(JROF,JLEV,3)=ILO3(JROF)+YDSL%NASLB1*ILEV
    ENDDO
    ! * Mask calculation for on-demand communications:
   IF(NPROC > 1.AND.YDSL%LSLONDEM_ACTIVE)THEN
!CDIR NODEP
!DIR$ PREFERVECTOR
     DO JROF=KST,KPROF
       YDSL%MASK_SL2(ILO (JROF)-IDIF  )=1
       YDSL%MASK_SL2(ILO (JROF)-IDIF+1)=1
       YDSL%MASK_SL2(ILO (JROF)-IDIF+2)=1
       YDSL%MASK_SL2(ILO (JROF)-IDIF+3)=1
       YDSL%MASK_SL2(ILO1(JROF)-IDIF  )=1
       YDSL%MASK_SL2(ILO1(JROF)-IDIF+1)=1
       YDSL%MASK_SL2(ILO1(JROF)-IDIF+2)=1
       YDSL%MASK_SL2(ILO1(JROF)-IDIF+3)=1
       YDSL%MASK_SL2(ILO2(JROF)-IDIF  )=1
       YDSL%MASK_SL2(ILO2(JROF)-IDIF+1)=1
       YDSL%MASK_SL2(ILO2(JROF)-IDIF+2)=1
       YDSL%MASK_SL2(ILO2(JROF)-IDIF+3)=1
       YDSL%MASK_SL2(ILO3(JROF)-IDIF  )=1
       YDSL%MASK_SL2(ILO3(JROF)-IDIF+1)=1
       YDSL%MASK_SL2(ILO3(JROF)-IDIF+2)=1
       YDSL%MASK_SL2(ILO3(JROF)-IDIF+3)=1
     ENDDO
   ENDIF
 ENDDO
```

BDW-LASCAW investigation

```
ILO(JROF)=IADDR_(IJ_,ILA  )+YDSL%NSLEXT(ILO(JROF) ,ILA  )
ILO1(JROF)=IADDR_(IJ_,ILA+1)+YDSL%NSLEXT(ILO1(JROF),ILA+1)
ILO2(JROF)=IADDR_(IJ_,ILA+2)+YDSL%NSLEXT(ILO2(JROF),ILA+2)
ILO3(JROF)=IADDR_(IJ_,ILA+3)+YDSL%NSLEXT(ILO3(JROF),ILA+3)
 DO JROF=KST,KPROF
      YDSL%MASK_SL2(ILO (JROF)-IDIF  )=1
      YDSL%MASK_SL2(ILO (JROF)-IDIF+1)=1
      YDSL%MASK_SL2(ILO (JROF)-IDIF+2)=1
      YDSL%MASK_SL2(ILO (JROF)-IDIF+3)=1
      YDSL%MASK_SL2(ILO1(JROF)-IDIF  )=1
      YDSL%MASK_SL2(ILO1(JROF)-IDIF+1)=1
      YDSL%MASK_SL2(ILO1(JROF)-IDIF+2)=1
      YDSL%MASK_SL2(ILO1(JROF)-IDIF+3)=1
      YDSL%MASK_SL2(ILO2(JROF)-IDIF  )=1
      YDSL%MASK_SL2(ILO2(JROF)-IDIF+1)=1
      YDSL%MASK_SL2(ILO2(JROF)-IDIF+2)=1
      YDSL%MASK_SL2(ILO2(JROF)-IDIF+3)=1
      YDSL%MASK_SL2(ILO3(JROF)-IDIF  )=1
      YDSL%MASK_SL2(ILO3(JROF)-IDIF+1)=1
      YDSL%MASK_SL2(ILO3(JROF)-IDIF+2)=1
      YDSL%MASK_SL2(ILO3(JROF)-IDIF+3)=1
ENDDO
```

```
ILOIK(JROF,JLEV)=IADDR(ILAV(JROF,JLEV)  )+YDSL%NSLEXT(ILOIK(JROF,JLEV) ,ILAV(JROF,JLEV) )
ILO1IK(JROF,JLEV)=IADDR(ILAV(JROF,JLEV)+1)+YDSL%NSLEXT(ILO1IK(JROF,JLEV),ILAV(JROF,JLEV)+1)
ILO2IK(JROF,JLEV)=IADDR(ILAV(JROF,JLEV)+2)+YDSL%NSLEXT(ILO2IK(JROF,JLEV),ILAV(JROF,JLEV)+2)
ILO3IK(JROF,JLEV)=IADDR(ILAV(JROF,JLEV)+3)+YDSL%NSLEXT(ILO3IK(JROF,JLEV),ILAV(JROF,JLEV)+3)
```

```
DO JROF=KST,KPROF
!DIR$ PREFERVECTOR
  DO JJ=0,3
    YDSL%MASK_SL2(ILOIK (JROF,JLEV)-IDIF+JJ)=1
    YDSL%MASK_SL2(ILO1IK(JROF,JLEV)-IDIF+JJ)=1
    YDSL%MASK_SL2(ILO2IK(JROF,JLEV)-IDIF+JJ)=1
    YDSL%MASK_SL2(ILO3IK(JROF,JLEV)-IDIF+JJ)=1
  ENDDO
 ENDDO
```

BDW-LASCAW results

Before Optimization:

Wall-times over all MPI-tasks (secs) : Avg=231.552, StDev=0.326

Avg-%	Avg.time	Min.time	Max.time	St.dev	Imbal-%	# of calls : Name of the routine
3.94%	9.111	7.876	10.058	0.961	21.69%	2014200 : LASCAW

After Optimization:

Wall-times over all MPI-tasks (secs) : Avg=227.252, StDev=0.511

Routines whose total time (i.e. sum) > 1.000 secs will be included in the listing

Avg-%	Avg.time	Min.time	Max.time	St.dev	Imbal-%	# of calls : Name of the routine
2.39%	5.420	4.972	5.698	0.335	12.74%	2014200 : LASCAW

LASCAW now moves from 3rd most time consuming to 8th

LASCAW is now 41.6% faster.

Saving us more than 3.7s in total runtime.

IFS new profile

- Broadwell 36 core optimized run gives following TOP 10:

Number of MPI-tasks : 6
Number of OpenMP-threads : 6
Wall-times over all MPI-tasks (secs) : Avg=223.890, StDev=0.395
Routines whose total time (i.e. sum) > 1.000 secs will be included in the listing

{ 8s absolute time } { 3.4% Faster overall }

Avg-%	Avg.time	Min.time	Max.time	St.dev	Imbal-%	# of calls : Name of the routine
8.07%	18.055	17.705	18.499	0.281	4.29%	805680 : CLOUDSC
3.92%	8.764	8.294	8.995	0.256	7.79%	367992173 : CUADJTQ
2.96%	6.628	6.241	7.467	0.530	16.42%	10071000 : LAITLI
2.79%	6.254	6.009	6.462	0.187	7.01%	402840 : CPG
2.68%	5.990	5.783	6.292	0.218	8.09%	2014200 : LARCHE
2.64%	5.909	5.301	6.327	0.382	16.22%	402840 : CLOUDVAR
2.52%	5.633	5.571	5.698	0.052	2.23%	402840 : CALLPAR
2.42%	5.428	4.981	5.729	0.336	13.06%	2014200 : LASCAW
2.35%	5.270	4.771	6.285	0.708	24.09%	2014200 : LAITRI
2.28%	5.103	2.200	8.551	2.775	74.27%	13140 : >OMP-TRLTOGPACK(1605)

KNL vs BDW original profile

- **KNL 68 core run gives following TOP 10+3:**

Number of MPI-tasks : 68

Number of OpenMP-threads : 4

Wall-times over all MPI-tasks (secs) : Avg=239.752, StDev=1.122

Routines whose total time (i.e. sum) > 1.000 secs will be included in the listing

Avg-%	Avg.time	Min.time	Max.time	St.dev	Imbal-%	# of calls : Name of the routine	vs BDW run: Without Opts	vs BDW run: With Opts
11.32%	27.129	25.853	28.931	0.682	10.64%	807840 : CLOUDSC	↑ Was 8.96%	Was 8.07%
4.63%	11.088	8.332	12.312	0.672	32.33%	371946456 : CUADJTQ		
3.60%	8.636	7.474	9.858	0.543	24.18%	46240 : >OMP-PHYSICS(1001)		
3.22%	7.707	6.636	9.580	0.671	30.73%	2019600 : LARCHE		
2.75%	6.598	1.037	14.035	3.139	92.61%	24820 : >MPL-TRGTOL_COMMS(803)		
2.55%	6.112	5.940	6.248	0.074	4.93%	403920 : CALLPAR		
2.47%	5.917	3.904	7.201	0.799	45.79%	403920 : CUBASEN		
2.41%	5.779	5.572	6.250	0.125	10.85%	10098000 : LAITLI		
2.18%	5.216	2.431	7.852	1.083	69.04%	403920 : CLOUDVAR		
1.95%	4.668	4.532	5.071	0.085	10.63%	4443120 : VERINT_DGEMM_1		
1.90%	4.557	3.617	5.003	0.257	27.70%	403920 : CPG	↓ Was 4.49%	Was 2.79%
1.83%	4.385	4.227	4.685	0.085	9.78%	2019600 : LAITRI		
1.68%	4.016	3.625	4.264	0.112	14.99%	2019600 : LASCAW	↓ Was 3.94%	Was 2.42%

KNL optimizations result

- CLOUDSC:

Only 9 initial “potential hazard”

Almost identical performance to non-optimized

- CPG:

Vectorizing the outer loop slowed down CPG

New result was 2.12% vs old 1.90%

- LASCAW:

Due to faster memory we still see speedup but less.

New result was 1.55% vs 1.72%

Thank You!

Especially:

Deborah Salmond
Sami Saarinen

Questions?