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:

```bash
#!/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:

```bash
export PMI_NO_FORK=1
export LD_LIBRARY_PATH=sdelib_path:$LD_LIBRARY_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 required 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]\)\(ADD\|SUB\)\(\d\d|\d\d\)\(PD\|SD\|SS\|PS\)\(_(X|Y|Z)\", line)
FLOP counting with Intel SDE (results on HPL)

- Run without SDE reports: **2316 GFLOPS**
- Run with SDE reports: **2297 GFLOPS**
- 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

| Counter Type                                      | Value               
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU_CLK_THREAD_UNHALTED:THREAD_P</td>
<td>1,148,950,914,774</td>
</tr>
<tr>
<td>CPU_CLK_THREAD_UNHALTED:REF_XCLK</td>
<td>49,610,954,365</td>
</tr>
<tr>
<td>DTLB_LOAD_MISSES:MISS_CAUSES_A_WALK</td>
<td>481,754,204</td>
</tr>
<tr>
<td>DTLB_STORE_MISSES:MISS_CAUSES_A_WALK</td>
<td>53,646,594</td>
</tr>
<tr>
<td>L1d:REPLACEMENT</td>
<td>71,972,039,189</td>
</tr>
<tr>
<td>L2_QSTS:ALL_DEMAND_DATA_RD</td>
<td>13,713,375,918</td>
</tr>
<tr>
<td>L2_QSTS:DEMAND_DATA_RD_HIT</td>
<td>7,179,862,200</td>
</tr>
<tr>
<td>MEM_UOPS_RETIRED:ALL_LOADS</td>
<td>787,829,823</td>
</tr>
<tr>
<td>FP_ARITH:SCALAR_DOUBLE</td>
<td>2,512,276,465</td>
</tr>
<tr>
<td>FP_ARITH:128B_PACKED_DOUBLE</td>
<td>432,030,907</td>
</tr>
<tr>
<td>FP_ARITH:256B_PACKED_DOUBLE</td>
<td>2,710,780,379,690</td>
</tr>
</tbody>
</table>

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

||Ax-b||_oo/(eps*(|A||_oo||x||_oo+||b||_oo)*N)= 0.0018303 ...... PASSED

21,422.43 (M/sec per task) x 72 (task)= 1.542e+03
Or else.... 99.74% accurate based on HPL output!
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

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

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:
  
  \[
  ZTMPA = 1.0 \_JPRB/ \text{MAX}(ZA(JL,JK),\text{ZEPESEC})
  \]

- 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:

869. IF (LDUSEPB1) THEN
870. + 1------< DO JFLD=1,NFLDSLBB1
871. 1 !DIR$ PREFERVECTOR
872. 1 !DIR$ IVDEP
873. 1 !DIR$ NOINTERCHANGE
874. + 1 Vpr2--< DO JROF=KSTGLO,MIN(KSTGLO-1+NPROMA,KGPCOMP)
875. 1 Vpr2 PB1(YDSL%NSLCORE(JROF),JFLD)=ZSLBUF1AU(JROF-KSTGLO+1,JFLD)
876. 1 Vpr2--> ENDDO
877. 1------> ENDDO
878. ENDF
BDW-CPG investigation

Vectorize by the outer loop:

869. IF (LDUSEPB1) THEN
870. Vb------< DO JFLD=1,NFLDSLB1
871. Vb !DIR$ PREFERVECTOR !DIR$ NEXTSCALAR
872. Vb !DIR$ IVDEP
873. Vb !DIR$ NOINTERCHANGE
874. Vb br4--< DO JROF=KSTGLO,MIN(KSTGLO-1+NPROMA,KGPCOMP)
875. Vb br4 PB1(YDSL%NSLCORE(JROF),JFLD)=ZSLBUF1AU(JROF-KSTGLO+1,JFLD)
876. Vb br4--> ENDDO
878. Vb------> ENDDO
879. ENDF
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,KLEV+KSLEV
  DO JROF=KST,KPROF
    IF (KHOR == 1) ILEV=KFLDN
    IF (KHOR == 0) ILEV=JLEV
    * Calculation of linear weights, KL0, KLH0.
    IF (LLCOMADH) THEN
      ZZWH(JROF,3,JLEV)=(ZDA*ZDB)*ZDC*ZRIPI2_(IJ_,ILA+1)
      ZZWH(JROF,2,JLEV)=(ZDA*ZDB)*ZDD*ZRIPI1_(IJ_,ILA+1)
      ZZWH(JROF,1,JLEV)=(ZDA*ZDC)*ZDD*ZRIPI0_(IJ_,ILA+1)
      IILA(JROF,JLEV)=ILA
      ZDD = PLAT(JROF,JLEV)
      ZDC = PLAT(JROF,JLEV)
      ZDB = PLAT(JROF,JLEV)
      ZDA = PLAT(JROF,JLEV)
    ELSE
      IF (LLCOMADV) THEN
        ZZWHMAD(JROF,3,JLEV)=ZZWH(JROF,3,JLEV)
        ZZWHMAD(JROF,2,JLEV)=ZZWH(JROF,2,JLEV)
        ZZWHMAD(JROF,1,JLEV)=ZZWH(JROF,1,JLEV)
        PDLAMAD(JROF,JLEV)=PDLAT(JROF,JLEV)
      ENDIF
      ZZWHMAD(JROF,3,JLEV)=(ZDA*ZDB)*ZDC*ZRIPI2_(IJ_,ILA+1)
      ZZWHMAD(JROF,2,JLEV)=(ZDA*ZDB)*ZDD*ZRIPI1_(IJ_,ILA+1)
      ZZWHMAD(JROF,1,JLEV)=(ZDA*ZDC)*ZDD*ZRIPI0_(IJ_,ILA+1)
      PDLAMAD(JROF,JLEV)=PDLAT(JROF,JLEV)*ZSTDDISV(JROF,JLEV) + 0.5_JPRB * ZSTDDISV(JROF,JLEV)
    ENDIF
    ILO(JROF)=IADDR_(IJ_,ILA )+YDSL%NSLEXT(ILO(JROF),ILA )
    ENDIF
    ELSE
      IF (LLCOMADH) THEN
        ZZWHMAD(JROF,3,JLEV)=ZZWH(JROF,3,JLEV)
        ZZWHMAD(JROF,2,JLEV)=ZZWH(JROF,2,JLEV)
        ZZWHMAD(JROF,1,JLEV)=ZZWH(JROF,1,JLEV)
        PDLAMAD(JROF,JLEV)=PDLAT(JROF,JLEV)
      ENDIF
      ZZWHMAD(JROF,3,JLEV)=(ZDA*ZDB)*ZDC*ZRIPI2_(IJ_,ILA+1)
      ZZWHMAD(JROF,2,JLEV)=(ZDA*ZDB)*ZDD*ZRIPI1_(IJ_,ILA+1)
      ZZWHMAD(JROF,1,JLEV)=(ZDA*ZDC)*ZDD*ZRIPI0_(IJ_,ILA+1)
      PDLAMAD(JROF,JLEV)=PDLAT(JROF,JLEV)*ZSTDDISV(JROF,JLEV) + 0.5_JPRB * ZSTDDISV(JROF,JLEV)
    ENDIF
    ENDIF
  ENDDO
ENDDO

**ECMWF 17th Workshop**
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

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

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

<table>
<thead>
<tr>
<th>Avg-%</th>
<th>Avg.time</th>
<th>Min.time</th>
<th>Max.time</th>
<th>St.dev</th>
<th>Imbal-%</th>
<th># of calls : Name of the routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.94%</td>
<td>9.111</td>
<td>7.876</td>
<td>10.058</td>
<td>0.961</td>
<td>21.69%</td>
<td>2014200 : LASCAW</td>
</tr>
</tbody>
</table>

## 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

<table>
<thead>
<tr>
<th>Avg-%</th>
<th>Avg.time</th>
<th>Min.time</th>
<th>Max.time</th>
<th>St.dev</th>
<th>Imbal-%</th>
<th># of calls : Name of the routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.39%</td>
<td>5.420</td>
<td>4.972</td>
<td>5.698</td>
<td>0.335</td>
<td>12.74%</td>
<td>2014200 : LASCAW</td>
</tr>
</tbody>
</table>

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

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

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

<table>
<thead>
<tr>
<th>Avg-%</th>
<th>Avg.time</th>
<th>Min.time</th>
<th>Max.time</th>
<th>St.dev</th>
<th>Imbal-%</th>
<th># of calls</th>
<th>Name of the routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.32%</td>
<td>27.129</td>
<td>25.853</td>
<td>28.931</td>
<td>0.682</td>
<td>10.64%</td>
<td>807840</td>
<td>CLOUDSC</td>
</tr>
<tr>
<td>4.63%</td>
<td>11.088</td>
<td>8.332</td>
<td>12.312</td>
<td>0.672</td>
<td>32.33%</td>
<td>371946456</td>
<td>CUADJTQ</td>
</tr>
<tr>
<td>3.60%</td>
<td>8.636</td>
<td>7.474</td>
<td>9.858</td>
<td>0.543</td>
<td>24.18%</td>
<td>46240</td>
<td>&gt;OMP-PHYSICS(1001)</td>
</tr>
<tr>
<td>3.22%</td>
<td>7.707</td>
<td>6.636</td>
<td>9.580</td>
<td>0.671</td>
<td>30.73%</td>
<td>2019600</td>
<td>LARCHES</td>
</tr>
<tr>
<td>2.75%</td>
<td>6.598</td>
<td>1.037</td>
<td>14.035</td>
<td>3.139</td>
<td>92.61%</td>
<td>24820</td>
<td>&gt;MPL-TRGTOL_COMMS(803)</td>
</tr>
<tr>
<td>2.55%</td>
<td>6.112</td>
<td>5.940</td>
<td>6.248</td>
<td>0.074</td>
<td>4.93%</td>
<td>403920</td>
<td>CALLPAR</td>
</tr>
<tr>
<td>2.47%</td>
<td>5.917</td>
<td>3.904</td>
<td>7.201</td>
<td>0.799</td>
<td>45.79%</td>
<td>403920</td>
<td>CUBASEN</td>
</tr>
<tr>
<td>2.41%</td>
<td>5.779</td>
<td>5.572</td>
<td>6.250</td>
<td>0.125</td>
<td>10.85%</td>
<td>10098000</td>
<td>LAITLI</td>
</tr>
<tr>
<td>2.18%</td>
<td>5.216</td>
<td>2.431</td>
<td>7.852</td>
<td>1.083</td>
<td>69.04%</td>
<td>403920</td>
<td>CLOUDVAR</td>
</tr>
<tr>
<td>1.95%</td>
<td>4.668</td>
<td>4.532</td>
<td>5.071</td>
<td>0.085</td>
<td>10.63%</td>
<td>4443120</td>
<td>VERINT DGEMM_1</td>
</tr>
</tbody>
</table>

  - Was 8.96%  
  - Was 4.49%  
  - Was 3.94%  

  - Was 4.49%  
  - Was 2.79%  
  - Was 2.42%

---

Without Opts  | With Opts
---|---
Was 8.07% |
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?