Documentation of array dimension reordering to pdfemcode3

Changes

The order of the dimension of certain multi-dimensional arrays was changed to optimize memory acces. The arrays that were changed are used in the function step in femswe.f90 and therefore used multiple times depending on the step count. An example is eofv(: , : , :), which is used in the function „dualadvflx“, that is used twice during one step. The old version looks like this:

```
DO iv1 = 1, nv
        temp = 0.0d0
        DO ix = 1, neofv(iv1,igrid)
                ie1 = eofv(iv1,ix,igrid)
                temp = temp + f(ie1)*eofvin(iv1,ix,igrid)
        ENDDO
        df(iv1) = temp
ENDDO
```

The index ix is used to iterate over the second dimension. The order of the required memory is eofv(1,1,igrid) → eofv(1,2,igrid) → eofv(1,3,igrid) → etc. But the index, that is changed during the loop should always be the first dimension. Thus, the required entries of the array are aligned in the memory. This leads to the following code.

```
DO iv1 = 1, nv
        temp = 0.0d0
        DO ix = 1, neofv(iv1,igrid)
                ie1 = eofv(ix,iv1,igrid)
                temp = temp + f(ie1)*eofvin(ix,iv1,igrid)
        ENDDO
        df(iv1) = temp
ENDDO
```

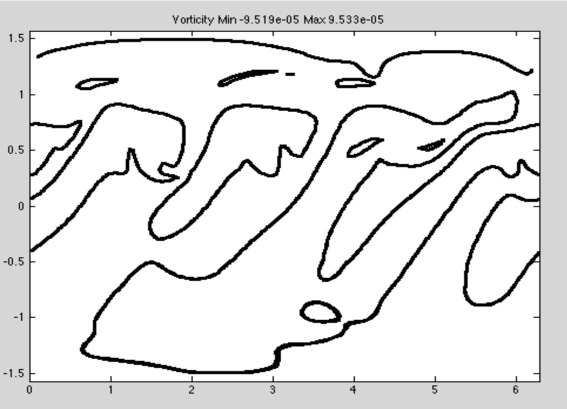
The only change is the exchange of the first and second dimension of the eofv-array. This was applied to all arrays that are used during the function step. The following list shows all modified arrays with an example, which array dimensions have been changed.

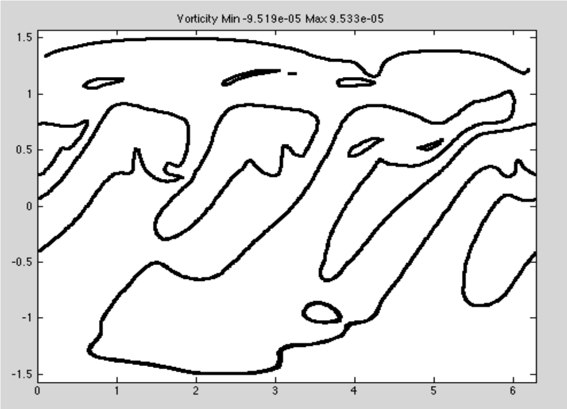
fnxte(if0, ix, igrid)	→	fnxte(ix, if0, igrid)
vofe(ie0, ix, igrid)	→	vofe(ix, ie0, igrid)
eoff(if0, ix, igrid)	→	eoff(ix, if0, igrid)
eoffin(if0, ix, igrid)	→	eoffin(ix, if0, igrid)
eofv(iv0, ix, igrid)	→	eofv(ix, iv0, igrid)
eofvin(iv0, ix, igrid)	→	eofvin(ix, iv0, igrid)
stenadvf(if0, ixs)	→	stenadvf(ixs, if0)
stenadvv(iv0, ixs)	→	stenadvv(ixs, iv0)
intmonf(if0, ixs, m)	→	intmonf(ixs, if0, m)
intmonv(iv0, ixs, m)	→	intmonv(ixs, iv0, m)
xminv(ie0, ixm, igrid)	→	xminv(ixm, ie0, igrid)
xminvsten(ie0, ixm, igrid)	→	xminvsten(ixm, ie0, igrid)
lsten(if0, ix, igrid)	→	lsten(ix, if0, igrid)
lmass(if0, ix, igrid)	→	lmass(ix, if0, igrid)
jstar(iv0, ix, igrid)	→	jstar(ix, iv0, igrid)
jsten(iv0, ix, igrid)	→	jsten(ix, iv0, igrid)
msten(ie0, ix, igrid)	→	msten(ix, ie0, igrid)
mmass(ie0, ix, igrid)	→	mmass(ix, ie0, igrid)
hstar(ie0, ix, igrid)	→	hstar(ix, ie0, igrid)
hsten(ie0, ix, igrid)	→	hsten(ix, ie0, igrid)
wsten(ie0, ix, igrid)	→	wsten(ix, ie0, igrid)
wcoeff(ie0, ix, igrid)	→	wcoeff(ix, ie0, igrid)
rxsten(iv0, ix, igrid)	→	rxsten(ix, iv0, igrid)
rxcoeff(iv0, ix, igrid)	→	rxcoeff(ix, iv0, igrid)
tsten(if0, ix, igrid)	→	tsten(ix, if0, igrid)
tcoeff(if0, ix, ixx, igrid)	→	tcoeff(ixx, ix, if0, igrid)
ressten(if0, ix, igrid)	→	ressten(ix, if0, igrid)
reswgt(if0, ix, igrid)	→	ressten(ix, if0, igrid)
injsten(if0, ix, igrid)	→	injsten(ix, if0, igrid)
injwgt(if0, ix, igrid)	→	injwgt(ix, if0, igrid)

Validation

To have correct results, these changes have to be done in the grid and operator generation as well. The restart files have been checked for bit-reproducibility. When using the GNU compiler (GCC version 4.8.2), all optimizations can be used. If the INTEL compiler (ifort version 13.1.3) is used, the program build_op has to be compiled without optimizations (-O0) to achieve bit-identical results.
The restart files are bit-identical for all test cases, that can be chosen in function „setini“.

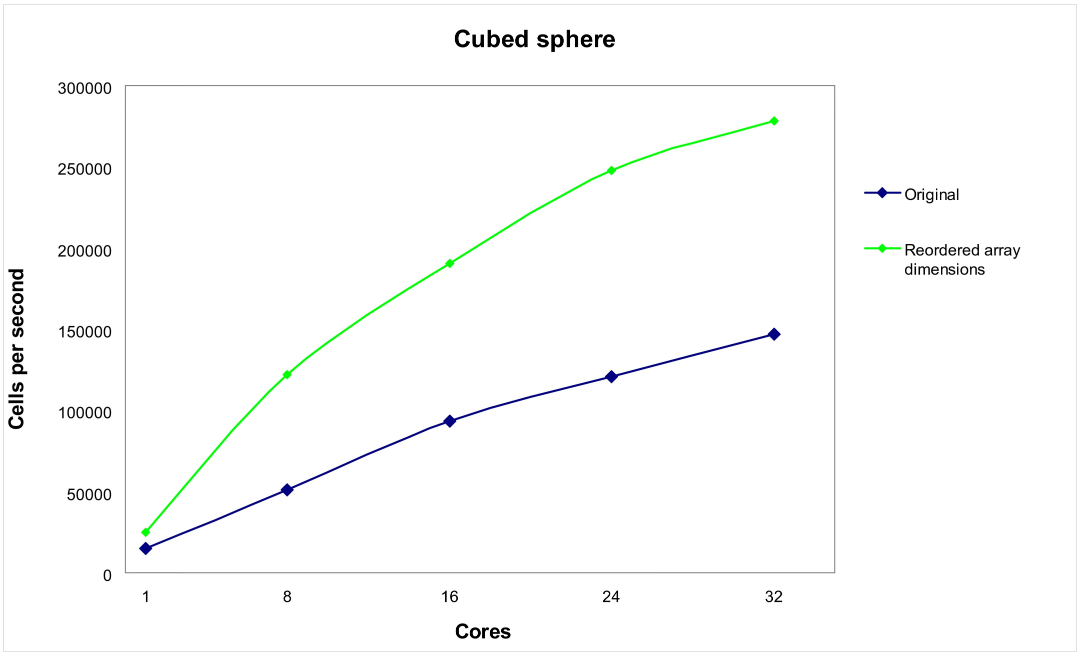
The following two pictures show the output, printed with matlab, for the Galewsky test after 144 hours (thus 288 time steps). On the left, it is the original version and on the right, we see the modified version. They are identical as well.





Results

The tests show a speed up of about a factor of 2 when using the reordered array dimension im comparison to the original version.



This speed up is possible due to the increased cache-hit rate on the L1-cache. The following data was collected with the perf tool for simulations on cubed sphere grids of level 8 (884736 faces).

Original version on 32 cores:

```
26,664,580,614,531 instructions
   377,388,612,718 cache-references
    71,676,995,307 cache-misses              #   18.993 % of all cache refs
 6,877,155,951,004 L1-dcache-loads
   434,306,228,677 L1-dcache-load-misses     #    6.32% of all L1-dcache hits
 1,671,973,967,882 L1-dcache-stores
    21,425,907,844 L1-dcache-store-misses
    46,315,660,545 L1-dcache-prefetches
    17,361,934,958 L1-dcache-prefetch-misses
   289,493,081,218 LLC-loads
   312,015,495,771 LLC-load-misses           #  107.78% of all LL-cache hits
   162,090,331,694 LLC-stores
    84,690,573,222 LLC-store-misses
    22,365,207,832 LLC-prefetches
    31,950,657,226 LLC-prefetch-misses

   742.303969857 seconds time elapsed
```

Modified version on 32 cores:

```
26,641,976,975,015 instructions
    51,551,063,908 cache-references
    35,631,508,474 cache-misses              #   69.119 % of all cache refs
 6,873,725,311,549 L1-dcache-loads
    95,381,514,832 L1-dcache-load-misses     #    1.39% of all L1-dcache hits
 1,655,546,676,918 L1-dcache-stores
    11,298,982,549 L1-dcache-store-misses
   100,285,399,166 L1-dcache-prefetches
    22,207,367,154 L1-dcache-prefetch-misses
    37,975,577,276 LLC-loads
    41,499,442,088 LLC-load-misses           #  109.28% of all LL-cache hits
    11,244,220,005 LLC-stores
     8,348,024,731 LLC-store-misses
    52,227,847,163 LLC-prefetches
    43,505,181,858 LLC-prefetch-misses
```

Original version on 1 core:

```
23,762,520,192,204 instructions
   488,569,935,962 cache-references
    47,333,125,281 cache-misses              #    9.688 % of all cache refs
 2,785,883,712,742 L1-dcache-loads
   343,780,267,096 L1-dcache-load-misses     #   12.34% of all L1-dcache hits
 1,139,736,888,897 L1-dcache-stores
    14,407,439,229 L1-dcache-store-misses
    19,308,156,504 L1-dcache-prefetches
    37,619,148,954 L1-dcache-prefetch-misses
   275,757,122,130 LLC-loads
   103,342,553,642 LLC-load-misses           #   37.48% of all LL-cache hits
    16,920,987,545 LLC-stores
    34,791,151,076 LLC-store-misses
   269,304,508,218 LLC-prefetches
   639,035,110,063 LLC-prefetch-misses

   6142.143991793 seconds time elapsed
```

Modified version on 1 core:

```
20,403,671,256,374 instructions
    66,783,900,233 cache-references
    44,940,410,376 cache-misses              #   67.292 % of all cache refs
 2,374,889,326,231 L1-dcache-loads
    81,991,345,449 L1-dcache-load-misses     #    3.45% of all L1-dcache hits
   975,117,139,093 L1-dcache-stores
    11,094,121,180 L1-dcache-store-misses
   176,924,583,557 L1-dcache-prefetches
    82,360,229,313 L1-dcache-prefetch-misses
    41,629,814,738 LLC-loads
   131,184,974,347 LLC-load-misses           #  315.12% of all LL-cache hits
    34,077,509,257 LLC-stores
    41,594,071,441 LLC-store-misses
   123,313,180,167 LLC-prefetches
   116,463,844,679 LLC-prefetch-misses
```