PD FEM CODE 3

General Info

Compiler Issues:

Intel Compiler

Compilation with Intel compiler and -O3 flag generates a binary *femswe* which is segfaulting for cubed sphere with 5 levels. => FIXED

The problem was the allocation of arrays with a given number of array entries on the subroutine's stack:

```
REAL*8 :: psi(nwhateverx)
```

This allocates the data directly on the thread's stack. Since this stack system is by far more limited compared to the system-wide memory, this segfaults for larger problem sizes since the thread is running out of stack size.

Solution: Dynamically allocate and deallocate large arrays, since this stores these arrays on the HEAP instead of the stack.

TODO: preallocate temporary arrays used during simulation time steps to avoid malloc/free overheads in subroutine *allocateall*

GNU Compiler:

Helps discovering out of boundary access or boundary mismatch detection with compile option *-fcheck=all .* Some lines were fixed with the help of this compiler option.

New Files:

adjacency_matrices.f90: Write out files with points of adjacency matrices and cell traversal

Usage: *./adjacency_matrices [cube/hex] [cells]*

The data can be plot with 'gnuplot' , generating png images, e.g.

$> gnuplot adjacencies_hex_0000010242.gnuplot

sfc_optimize.f90: Reorder cells along SFC curve

Usage: ./sfc_optimize [cube/hex] [cells]

Additional sfc index tables are appended to the gridmap files to lookup the “old” index.

F77 to F90

Converted gengrid_hex.f to gengrid_hex.f90

Bitreproducable with original code if using compile option -O0

Added Program arguments:

./gengrid_hex [levels]

./gengrid_cube [levels]

./buildfem_op [hex/cube] [cells]

./femswe [parameter file]

Multigrid Issues with SFC Optimization

The multi grid restriction operator assumes a particular cell indexing scheme (in *buildop_fem.f90*). After an SFC reordering, this scheme does not exist anymore. Therefore the sfc_optimize.f90 code writes additional lookup talbes to the grid-data files. These tables are then used to lookup the non-SFC-ordered position and the SFC-ordered position.

fNewFaceId: lookup table with old cell id stored at table entry.

fNewFaceIdInverse: lookup table with new cell id at table entry.

E.g. determining the cell for a MG prolongation operation on a hexagonal cell is then given by

fNewFaceIdInverse(fNewFaceId(if0, igrid), igridp), see buildop_fem.f90

Bit-wise reproducability

- The cell-ordering of the output files run_* had to be reconstructed based on *fNewFaceIdInverse*
- Bit-wise reproducable results generated for hex_10242 and cube_13824 without SFC optimization.

  With SFC optimization not yet reproducable due to some operations in the MG.

Implementation

- The SFC optimization program *sfc_optimize.f90* expects a grid-file as a data input and writes the SFC reordered cell-data to the same file or to an additional file if specified as a second program parameter.
- The reordering is based on computing the 1D SFC index and reordering all points. There's space for optimization regarding this ordering by using another SFC (Hilbert, Peano, Sierpinski, etc.):

  The function *computeSFCIndex* can be easily extended with other SFC curves.
- SFC 1D coordinate is computed with the Z-Curve [SUBROUTINE computeSFCIndex(X, o_scalar) in sfc_optimize.f90] based on the 3D cell's mid-point.
- See results: optimization for cells not enough and sometimes leads to less performance since the originally already existing locality between cells and edges/vertices is destroyed.

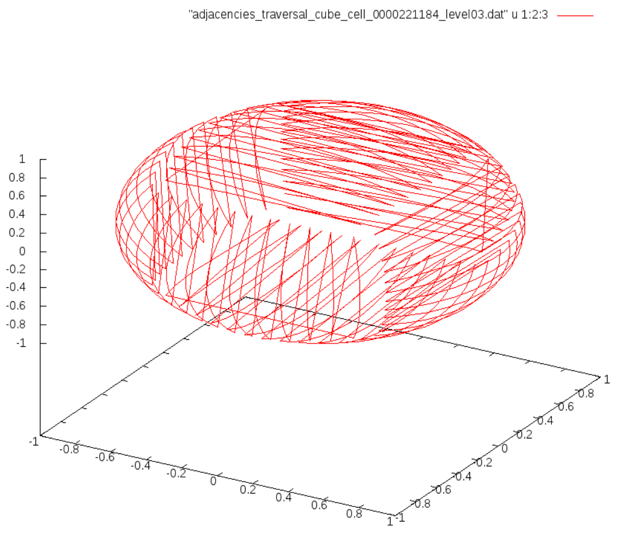
Cells/Edges/Nodes

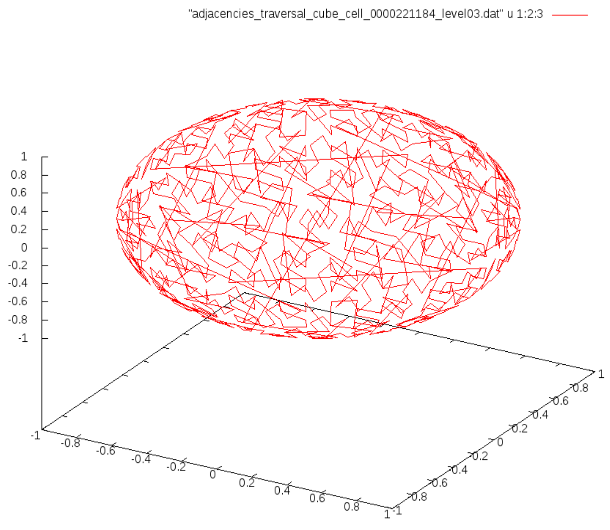
The SFC reordering was so far **only applied to the cells**.

Also applying it to edges and nodes, we expect performance enhancements.

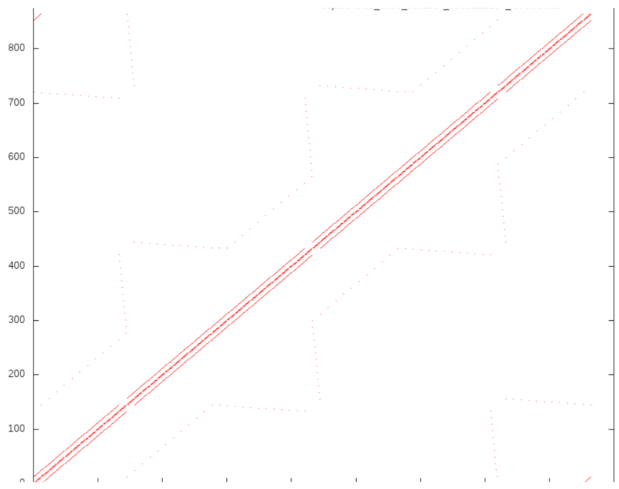
Cubed sphere:

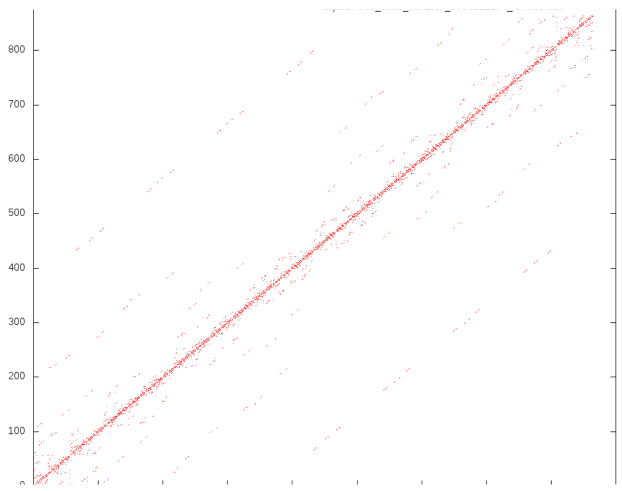
Displaying grid level 3 only



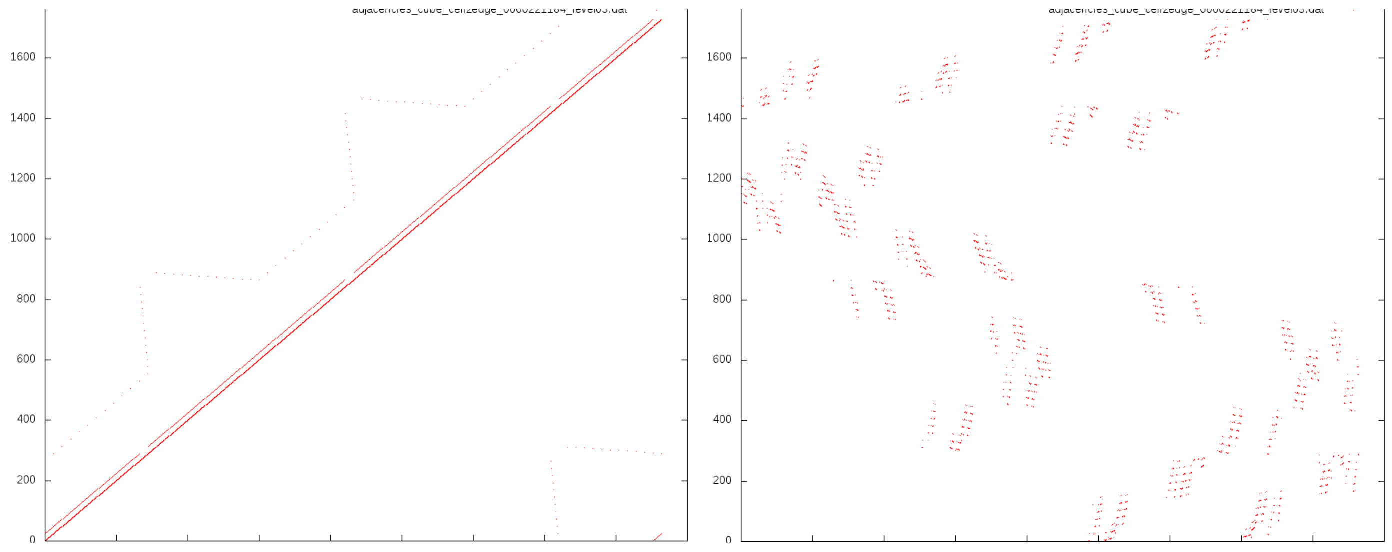


Left image: default ordering, right image: Z-ordered.





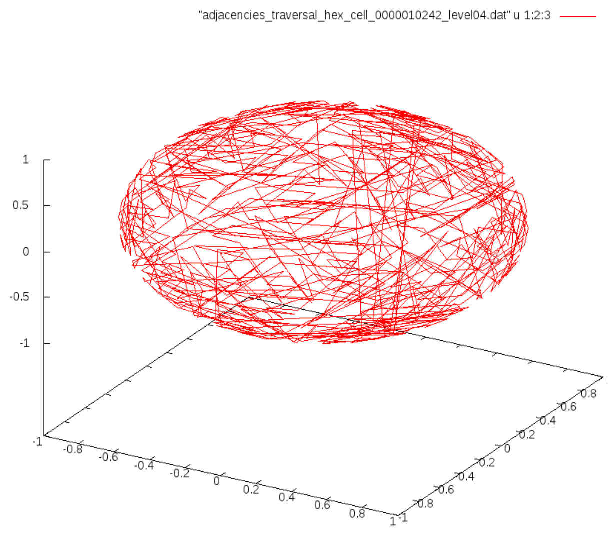
Left image: cell2cell adjacency matrix already almost diagonal. Right image: after ordering with the SFC.

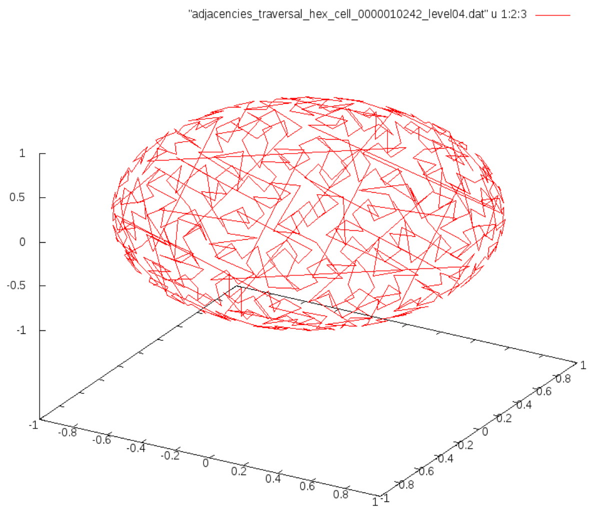


Reordering of cells “destroys” the edge-locality. Left image: Close-to diagonal cell-to-edge adjacency matrix. Right image: adjacency matrix after SFC-reordering

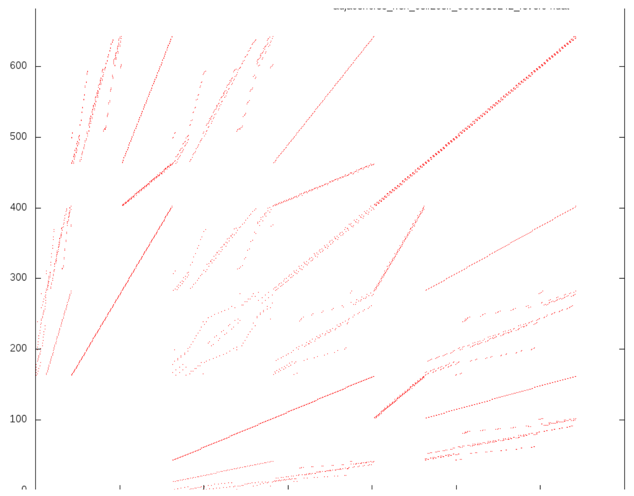
Hex-grids:

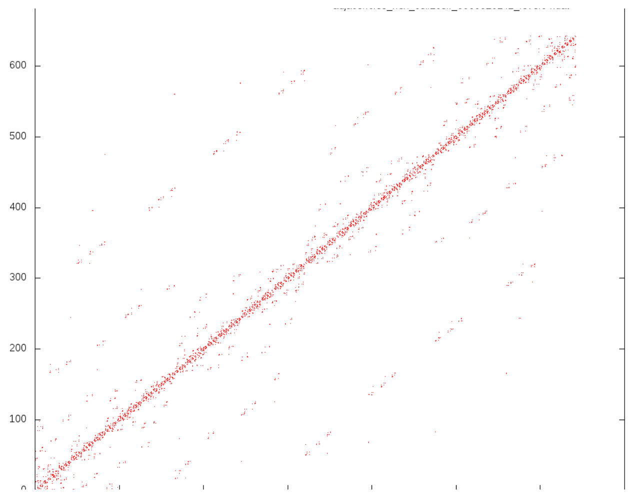
Info: Displaying grid-levels 4





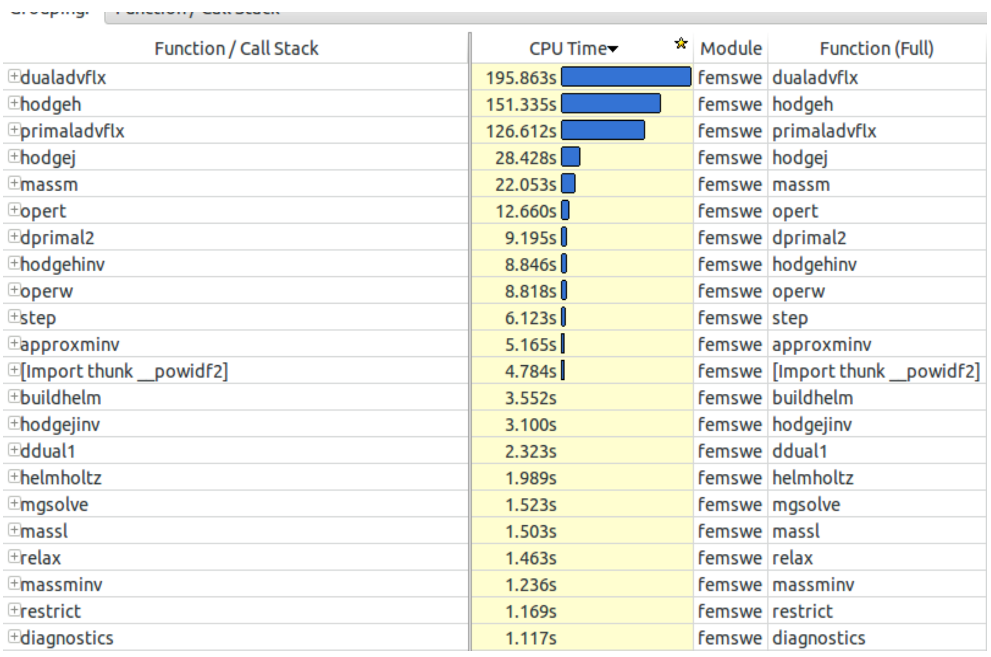
Left: default ordering of hex-grid. Right: Z-SFC ordered.





Left: Unordered cell2cell adjacency matrix, Right: SFC ordered

Vtune Amplifier:

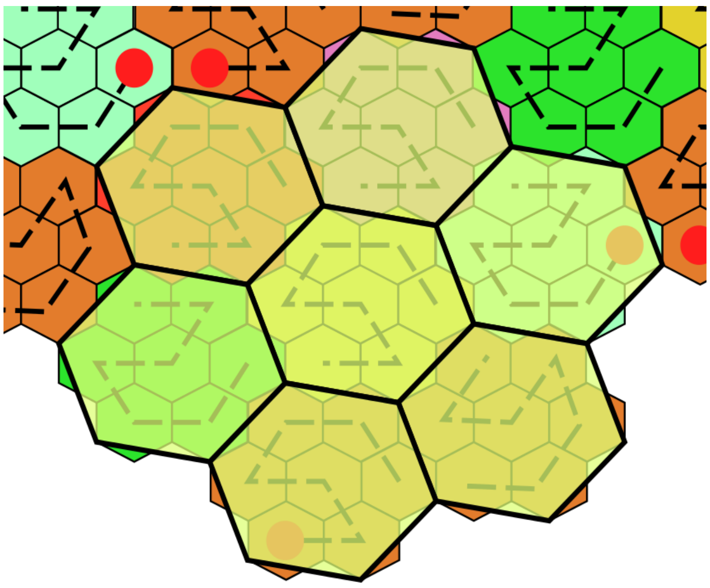


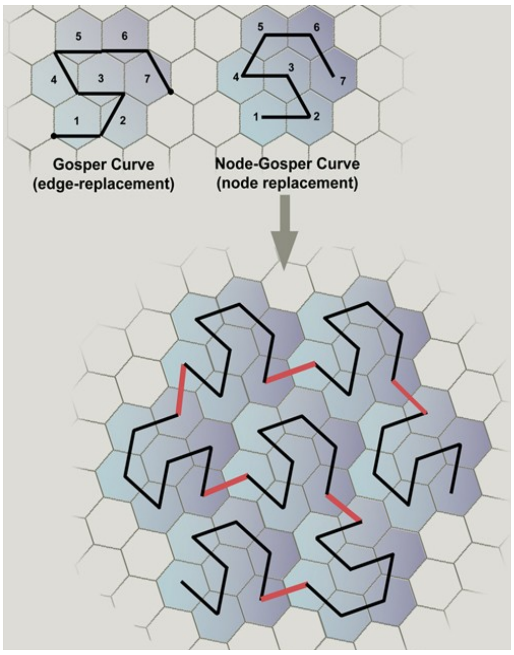
Function / Call Stack	CPU Time	Module	Function (Full)
dualadvflx	195.863s	femswe	dualadvflx
hodgeh	151.335s	femswe	hodgeh
primaladvflx	126.612s	femswe	primaladvflx
hodgej	28.428s	femswe	hodgej
massm	22.053s	femswe	massm
opert	12.660s	femswe	opert
dprimal2	9.195s	femswe	dprimal2
hodgehinv	8.846s	femswe	hodgehinv
operw	8.818s	femswe	operw
step	6.123s	femswe	step
approxminv	5.165s	femswe	approxminv
[Import thunk __powidf2]	4.784s	femswe	[Import thunk __powidf2]
buildhelm	3.552s	femswe	buildhelm
hodgejinv	3.100s	femswe	hodgejinv
ddual1	2.323s	femswe	ddual1
helmholtz	1.989s	femswe	helmholtz
mgsolve	1.523s	femswe	mgsolve
massl	1.503s	femswe	massl
relax	1.463s	femswe	relax
massminv	1.236s	femswe	massminv
restrict	1.169s	femswe	restrict
diagnostics	1.117s	femswe	diagnostics

Remarks and TODO

- SFC ordering for **edges** and **vertices** missing
- Optimize array access such as fnxtf(i,1:6,ngrids) to fnxtf(1:6,i,ngrids) since data accessed via index 1:6 is consecutively aligned in memory -> optimizations for cacheline.
- Implement & test other SFC such as Gosper curve for hexagonal grids.

(from http://courses.washington.edu/css342/fukuda/prog/prog3.html)





- Adaptive Gosper SFCs:

  Hanging nodes & non-matching edges & other ugly properties

Parallelization and Benchmarks

Source Code Files:

Parallelization done with OpenMP for femswe.f90 only.

Other codes have to be parallelized as well. Otherwise, no large-scale simulations can be executed.

Memory Consumption:

Info: Approx. 16 GB memory consumption for problem size with 655362 cells.

The reason for this excessive memory consumption can be e.g. found by allocating the same number of DOFs on each MG level. TODO: Further investigation needed.

Debugging:

Invalid OpenMP code detected with OpenMP debug version: Shared variables had to be private and were not optimized with OpenMP version => Always test with -O0, OpenMP and Intel.

Push/Pull access scheme

=> Replaced push with pull calls for restriction operator

Bit reproducability tested for Cubed sphere with 13824 cells.

Using workshare and parallel do parallelization:

- !$OMP PARALLEL DO DEFAULT(NONE) PRIVATE(iv0) SHARED(nvertx, psi, igrid)

  ```
  DO iv0 = 0, nvertx
      psi(iv0) = .........
  END DO
  !$OMP END PARALLEL DO
  ```

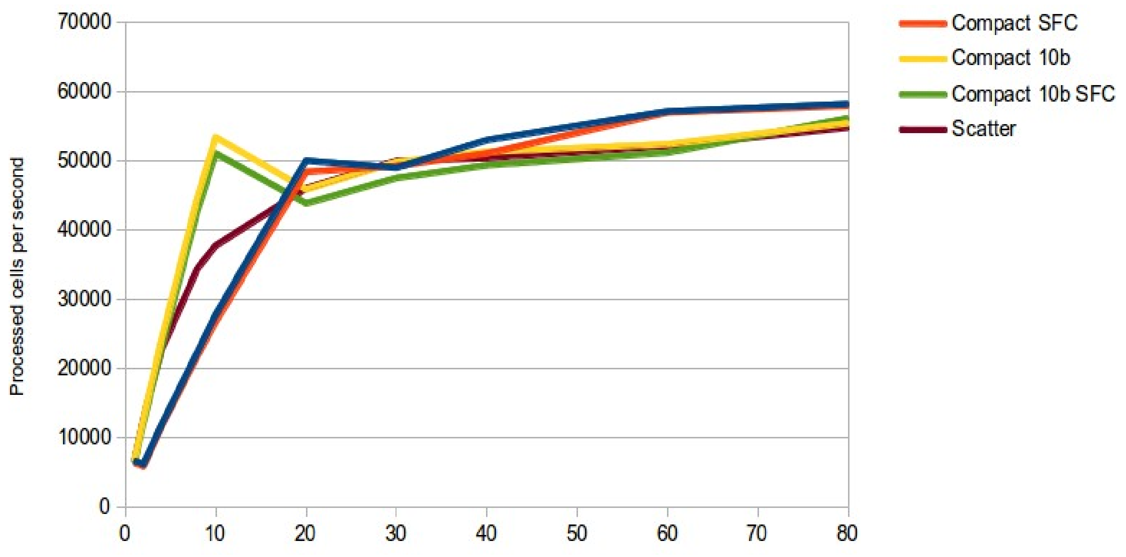
- Always use DEFAULT(NONE) to assure specification of PRIVATE or SHARED for all variables.

  PRIVATE: The storage for these variables is replicated for each thread, thus independent to other threads. E.g. local counters or array indices inside the loop have to be declared as PRIVATE.

  SHARED: The variables are directly accessed by the threads. Arrays should by typically declared as SHARED to avoid replicating them.

- Output of restart files and diagnostic computations deactivated.

- Benchmark:
  - Intel compiler environment variables
    - Scatter: export KMP_AFFINITY=scatter
    - Compact: export KMP_AFFINITY=compact (DO NOT USE THIS)
    - Compact 10: export KMP_AFFINITY=fine,compact,1,0
  - Westmere 4 Socket, 10 cores per Socket, hyperthreaded: 80 threads available.

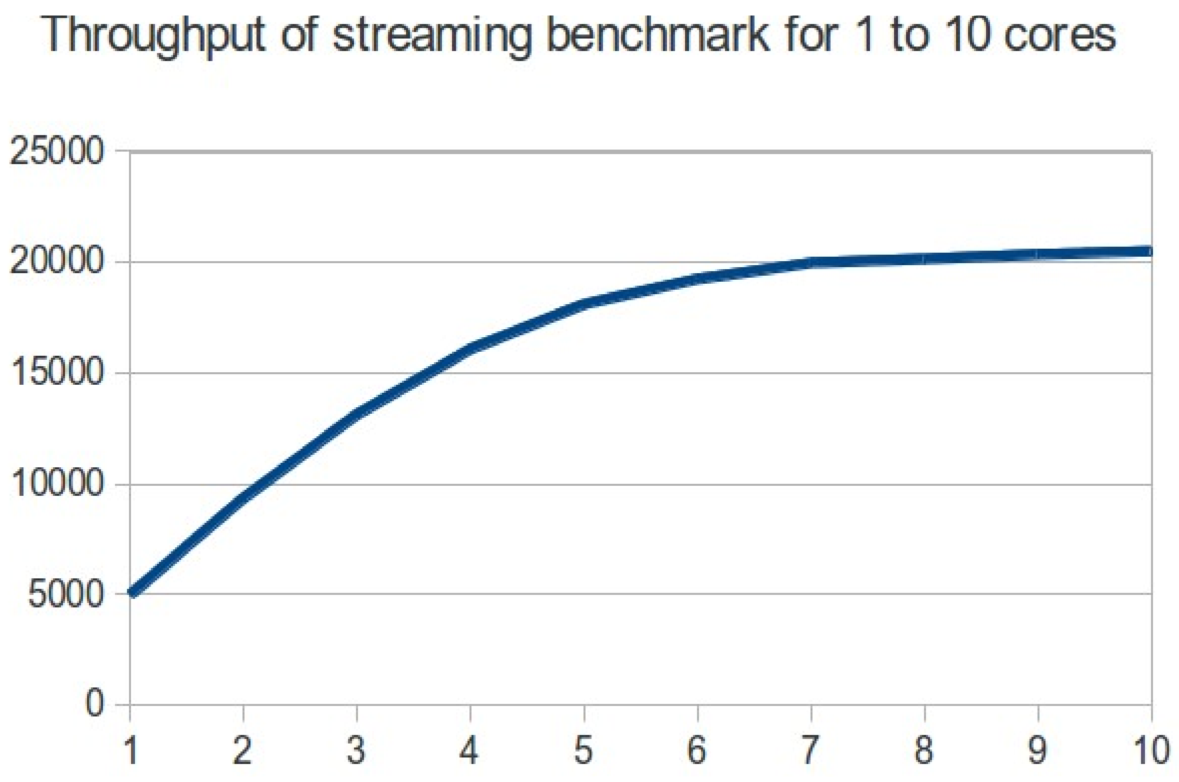


  - Scalability up to 10 cores.
  - No further scalability beyond 10 cores. Possible reasons:
    - First-touch policy on memory pages (close to bandwidth limited)?
    - Bandwidth limited?

      OR
    - Problem size is too small, overhead of parallelization too high?

Optimizing with first-touch policy for possibly memory-bandwidth limited problem

Example: streaming benchmark which copies data from one memory location to aother.



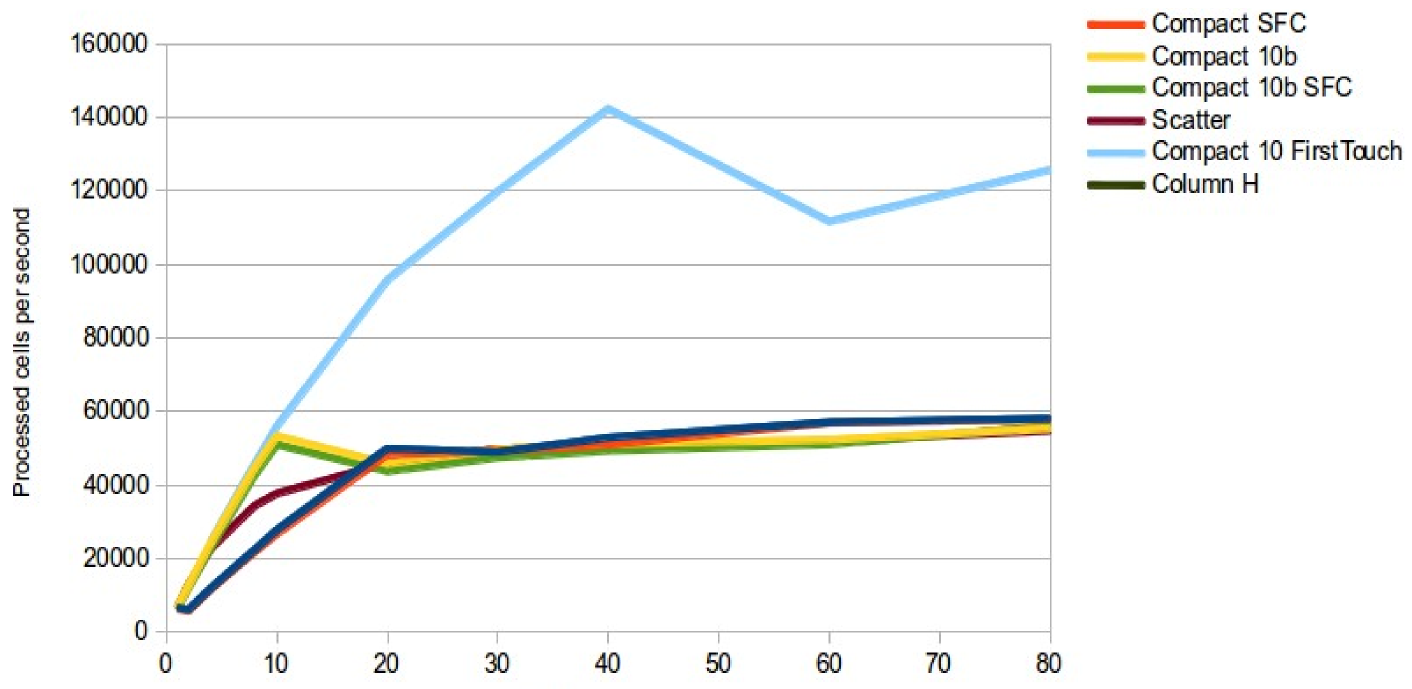
Scalability limited for bandwidth limited problems starting at ~6 cores.

For multiple CPUs, there are also multiple memory controller.

If allocating the memory with a single core (e.g. by initializing the data with the first core only), all data gets allocated at the memory assigned to a single memory controller.

First touch policy can help to overcome these issues:

Initialize the data using all cores in parallel to allocate memory at memory controllers with the assumption that the main memory access is also accomplished via the same memory controller for computing a simulation timestep.


Processed cells per second
Compact SFC
Compact 10b
Compact 10b SFC
Scatter
Compact 10 FirstTouch
Column H