Partitioning and numbering meshes for efficient MPI-parallel execution in PyOP2

Lawrence Mitchell, Mark Filipiak[1]

Tuesday 18th March 2013

[1] lawrence.mitchell@ed.ac.uk, mjf@epcc.ed.ac.uk

Outline

Numbering to be cache friendly

Numbering for parallel execution

Hybrid shared memory + MPI parallelisation

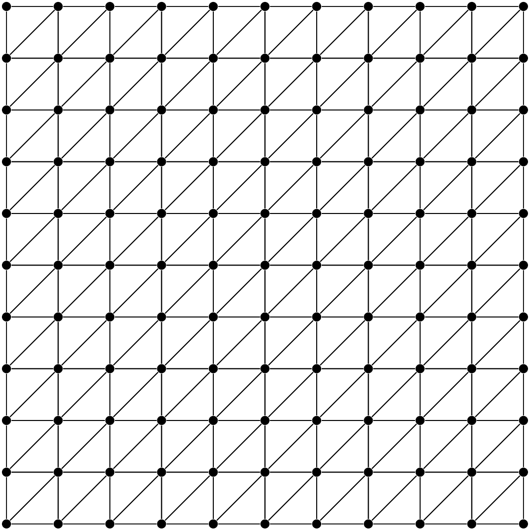
Modern hardware

- Latency to RAM is 100s of clock cycles
- Multiple caches to hide this latency
  - memory from RAM arrives in cache lines (64 bytes, 128 bytes on Xeon Phi)
  - hardware prefetching attempts to predict next memory access

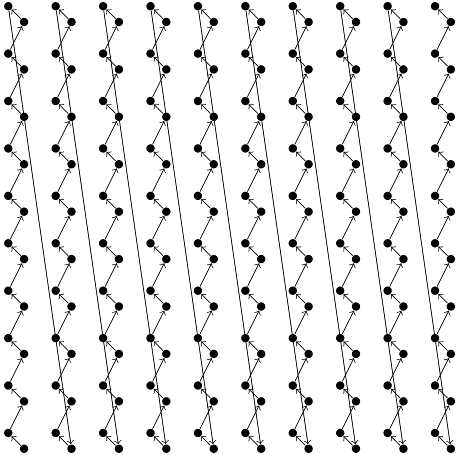
Exploiting hardware caches in FE assembly

- Direct loops over mesh entities are cache-friendly
- indirect loops may not be
  - can we arrange them to be cache friendly?

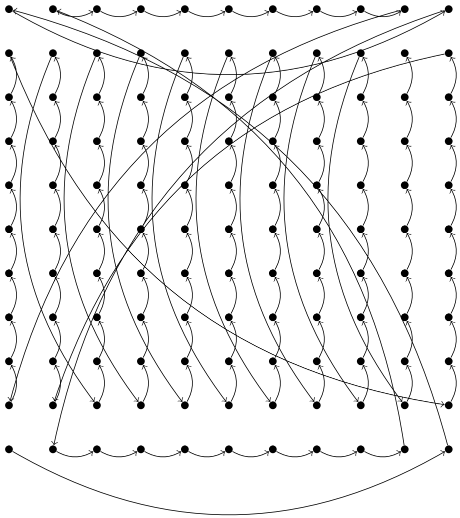
A mesh



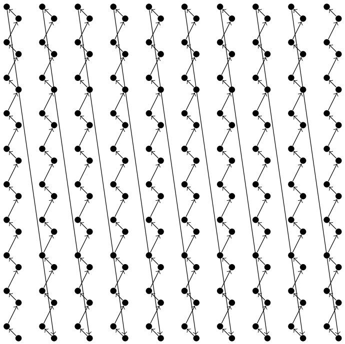
Cache friendly visit order (default numbering)

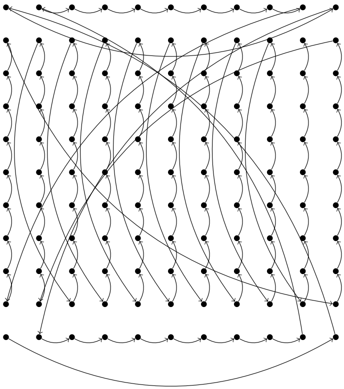


Cache friendly visit order (default numbering)



Cache friendly visit order (default numbering)



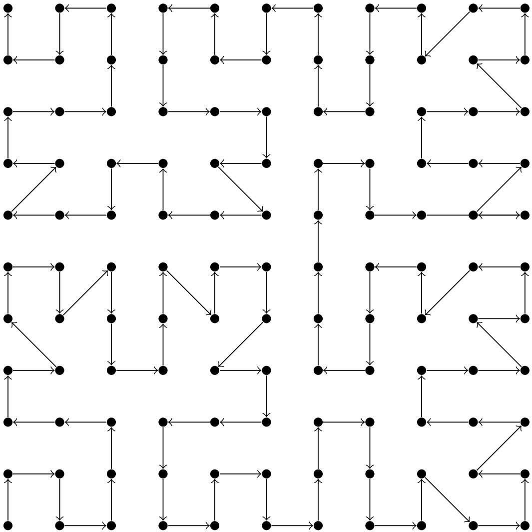


Mesh entity numbering is critical

- arrange for "connected" vertices to have a good numbering (close to each other)
- given this good numbering
  - derive numberings for other entities

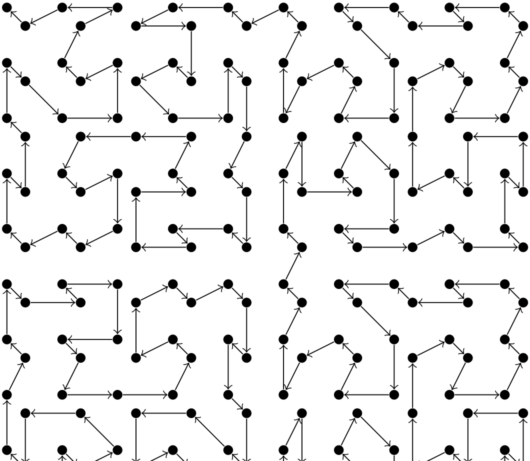
Numbering dofs

- Cover mesh with space-filling curve
  - vertices that are close to each other get close numbers

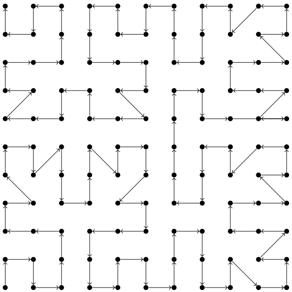


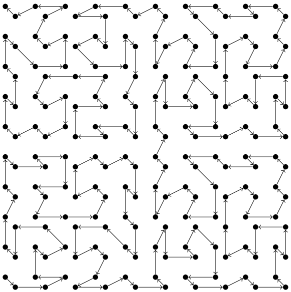
Other entities

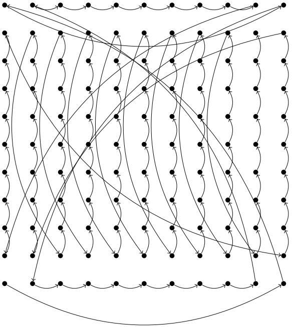
- construct additional entities with some numbering
- sort them and renumber lexicographically keyed on sorted list of vertices they touch
- do this every time the mesh topology changes
  - (doesn't work yet)

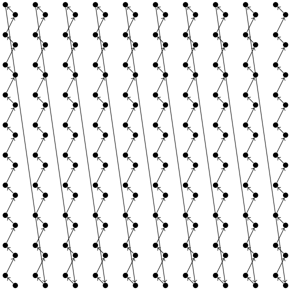


Comparing









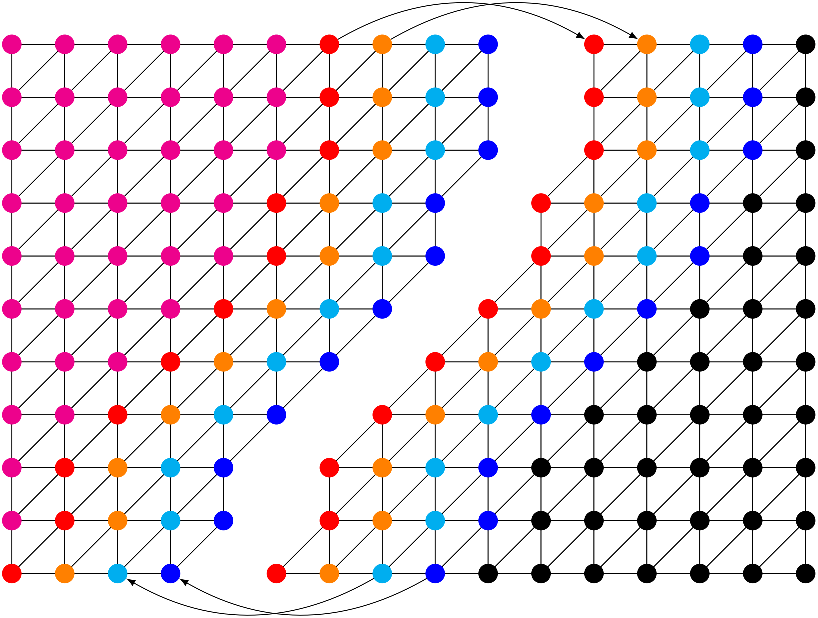
Does it work?

- In Fluidity
  - P1 problems get around 15% speedup
- In PyOP2
  - GPU/OpenMP backends get 2x-3x speedup (over badly numbered case)
  - Fluidity kernels provoke cache misses in other ways

Iteration in parallel

- Mesh distributed between MPI processes
- communicate halo data
- would like to overlap computation and communication

Picture



Comp/comms overlap

- entities that need halos can't be assembled until data has arrived
- can assemble the other entities already

```
start_halo_exchanges()
for e in entities:
    if can_assemble(e):
        assemble(e)
finish_halo_exchanges()
for e in entities:
    if still_needs_assembly(e):
        assemble(e)
```

Making this cheap

- separate mesh entities into groups

```
start_halo_exchanges()
for e in core_entities:
    assemble(e)
finish_halo_exchanges()
for e in additional_entities:
    assemble(e)
```

PyOP2 groups

- Core entities
  - can assemble these without halo data
- Owned entities
  - local, but need halo data
- Exec halo
  - off-process, but redundantly executed over (touch local dofs)
- Non-exec halo
  - off-process, needed to compute exec halo

Why like this?

- GPU execution
  - launch separate kernels for core and additional entities
  - no branching in kernel to check if entity may be assembled
- Defer halo exchange as much as possible (lazy evaluation)

How to separate the entities

- separate data structures for different parts
  - possible, but hurts direct iterations, and is complicated
- additional ordering constraint
  - core, owned, exec, non-exec
  - implemented in Fluidity/PyOP2
  - each type of mesh entity stored contiguously, obeying this ordering

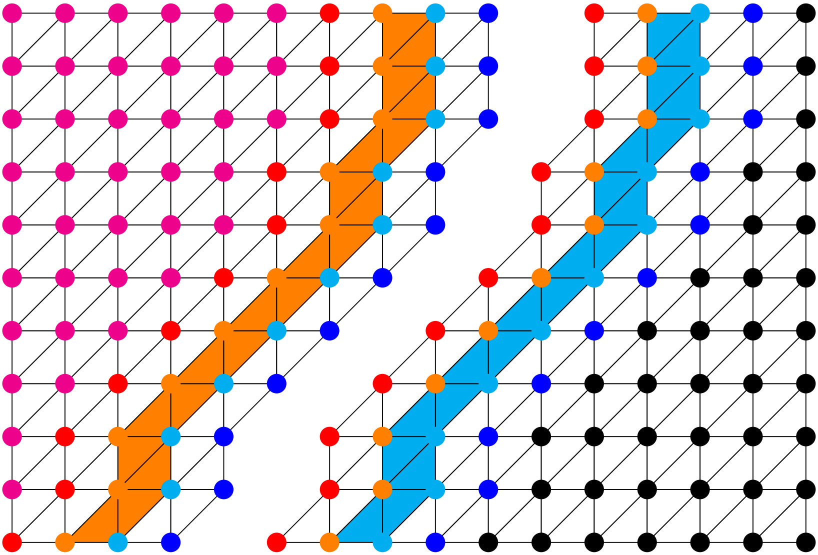
Hybrid shared memory + MPI parallelisation

- On boundary, assembling off-process entities can contribute to on-process dofs
- how to deal with this?
  - use linear algebra library that can deal with it
  - e.g. PETSc allows insertion and subsequent communication of off-process matrix and vector entries
- Not thread safe

Solution

- Do redundant computation
  - this is the default PyOP2 computation model
- Maintain larger halo
- assemble all entities that touch local dofs
  - turn off PETSc off-process insertion

Picture



Multiple gains

- You probably did the halo swap anyway
  - this makes form assembly non-communicating
- we've seen significant (40%) benefit on 1000s of processes (Fluidity only)
- thread safety!

Thread safety

- Concurrent insertion into MPI PETSc matrices *is* thread safe if:
  - there's no off-process insertion caching
  - user deals with concurrent writes to rows
    - colour the *local* sparsity pattern

Corollary

- It is possible to do hybrid MPI/OpenMP assembly with existing linear algebra libraries
  - implemented (and tested!) in PyOP2
- Ongoing work to add more shared memory parallisation in kernels in PETSc
  - PETSc team
  - Michael Lange (Imperial)

Conclusions

- With a bit a of work, we can make unstructured mesh codes reasonably cache friendly
- For good strong scaling, we'd like to overlap computation and communication as much as possible, but cheaply
- We think the approaches here work, and are implemented in Fluidity/PyOP2

Acknowledgements

- Hilbert reordering in Fluidity:
  - Mark Filipiak (EPCC) [a dCSE award from EPSRC/NAG]
- Lexicographic mesh entity numbering and ordering in Fluidity:
  - David Ham (Imperial), and me (prodding him along the way)
- PyOP2 MPI support:
  - me (EPCC) [EU FP7/277481 (APOS-EU)]
  - ideas from Mike Giles and Gihan Mudalige (Oxford)
- MAPDES team:
  - funding (EPSRC grant EP/I00677X/1, EP/I006079/1)