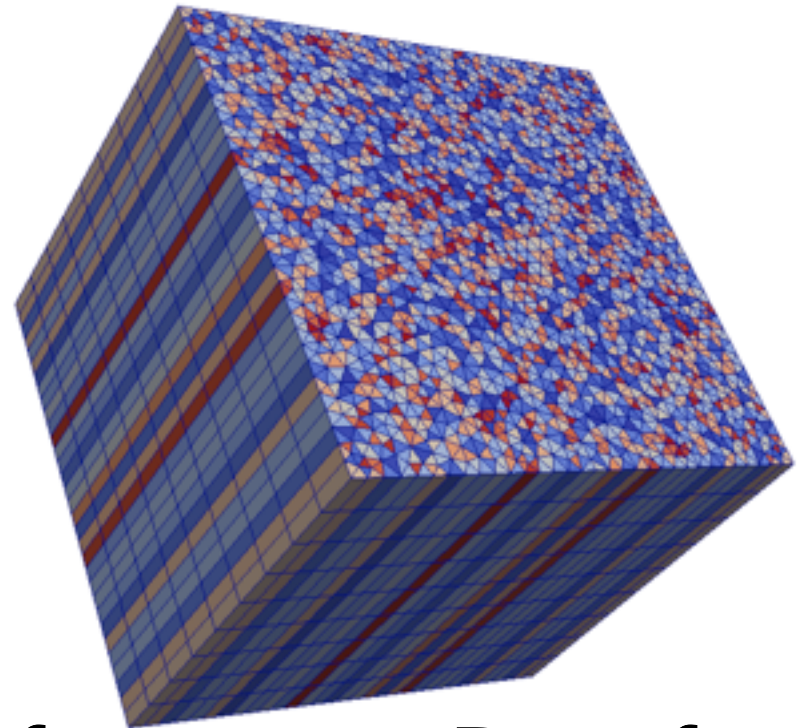
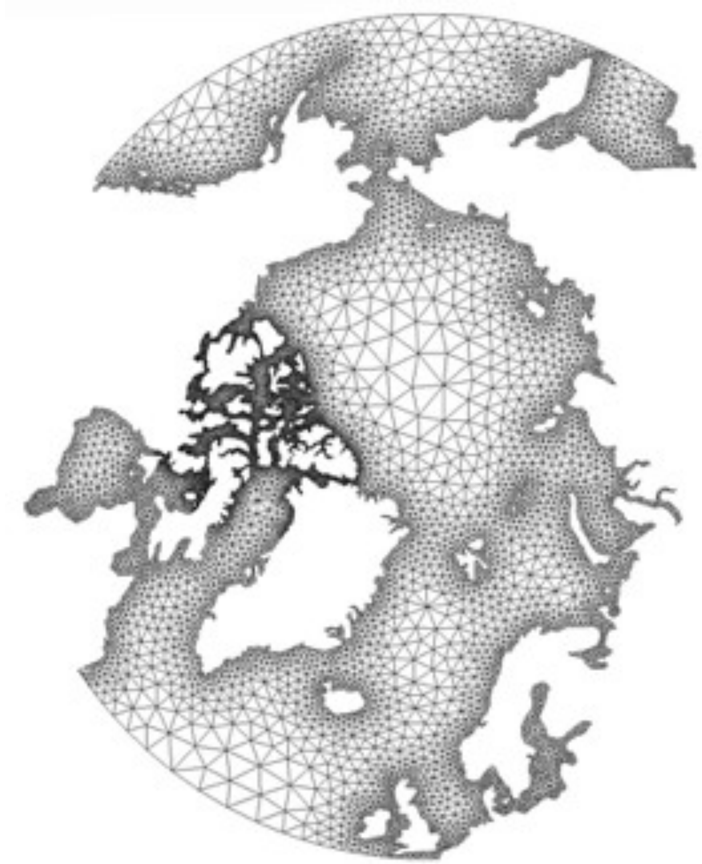


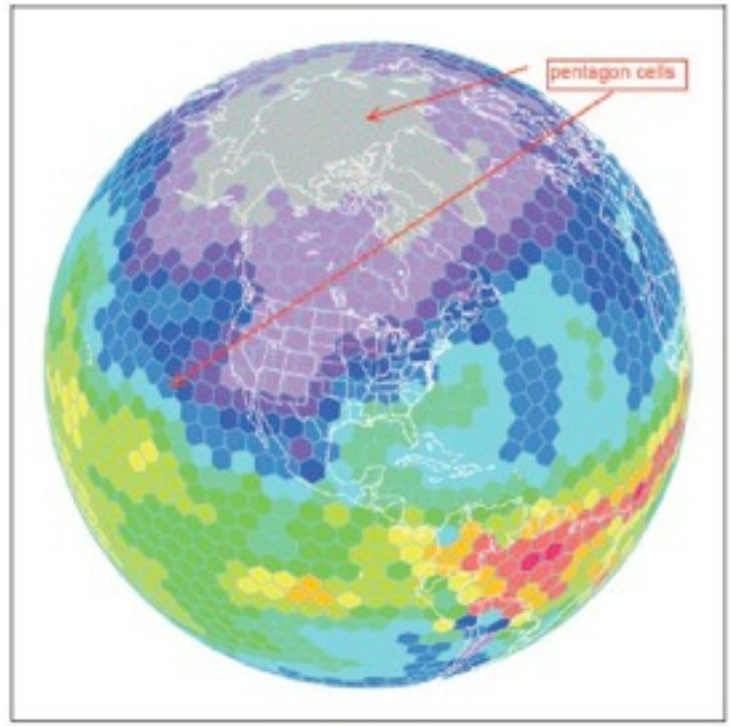
Exploiting Performance Benefits of Extruded Meshes in PyOP2

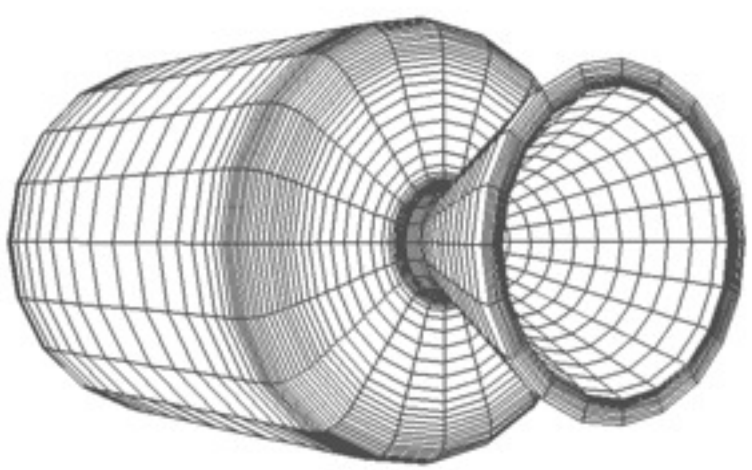
Department of Computing - Software Performance Optimisation Group
Imperial College London

Gheorghe-Teodor Bercea,
Florian Rathgeber, Fabio Luporini,
David A. Ham, Paul H. J. Kelly

Mesh-Based Simulation Applications



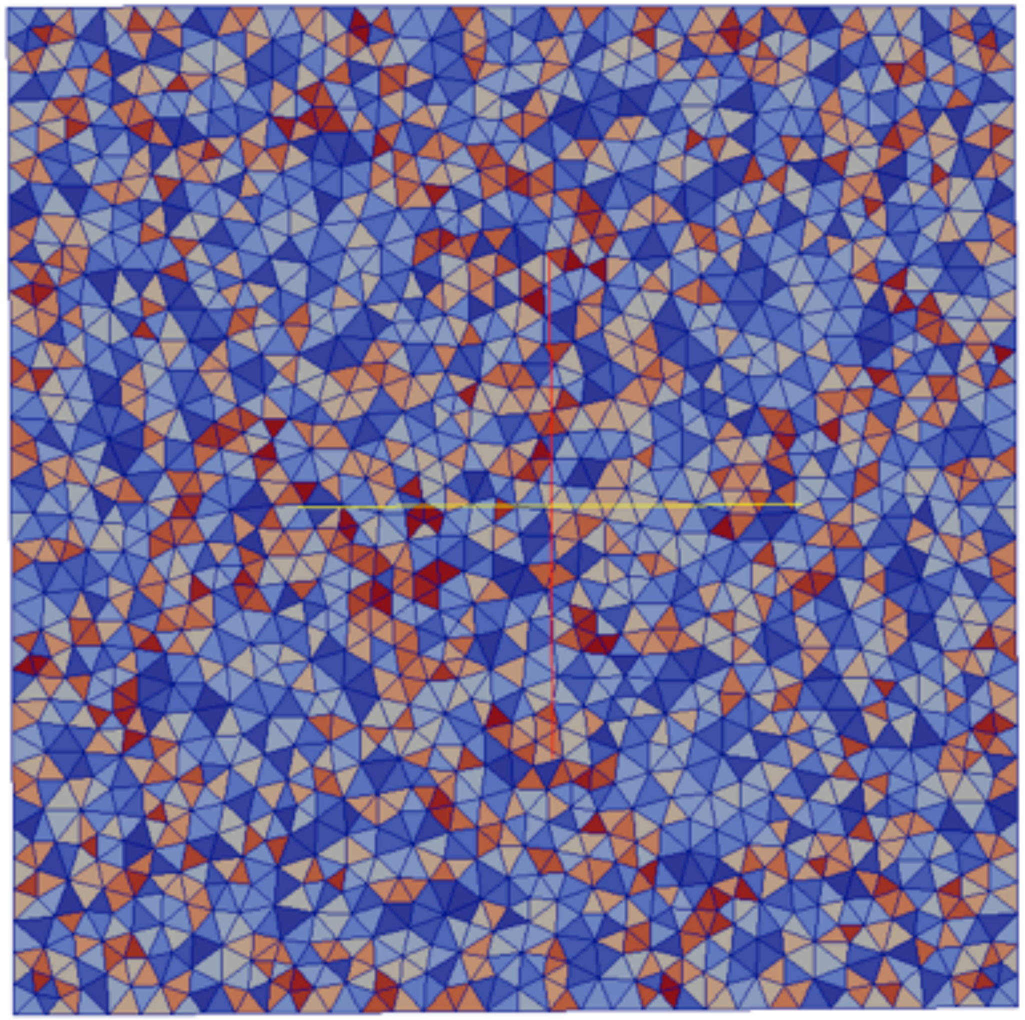


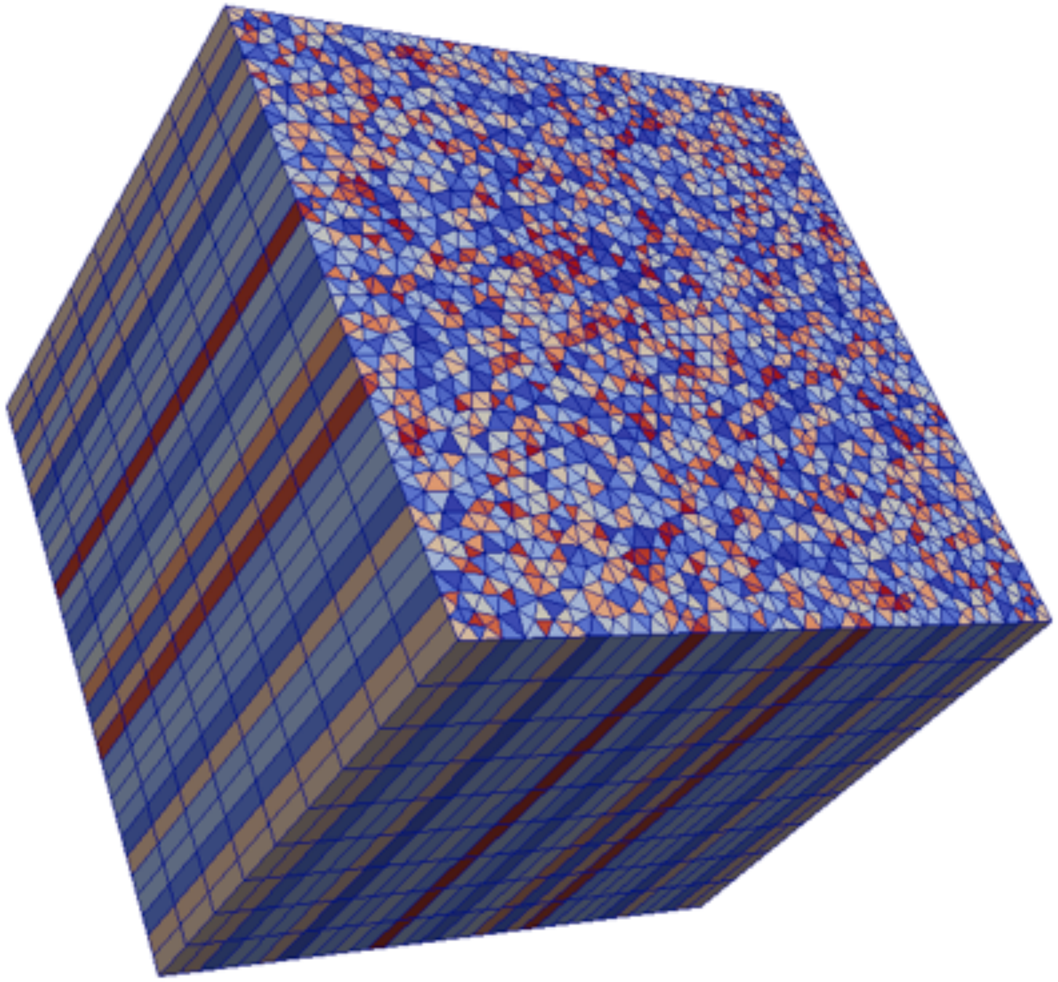


- Atmosphere and ocean modelling
- Climate models and numerical weather prediction
- Thin-shell object simulations

Types of Meshes

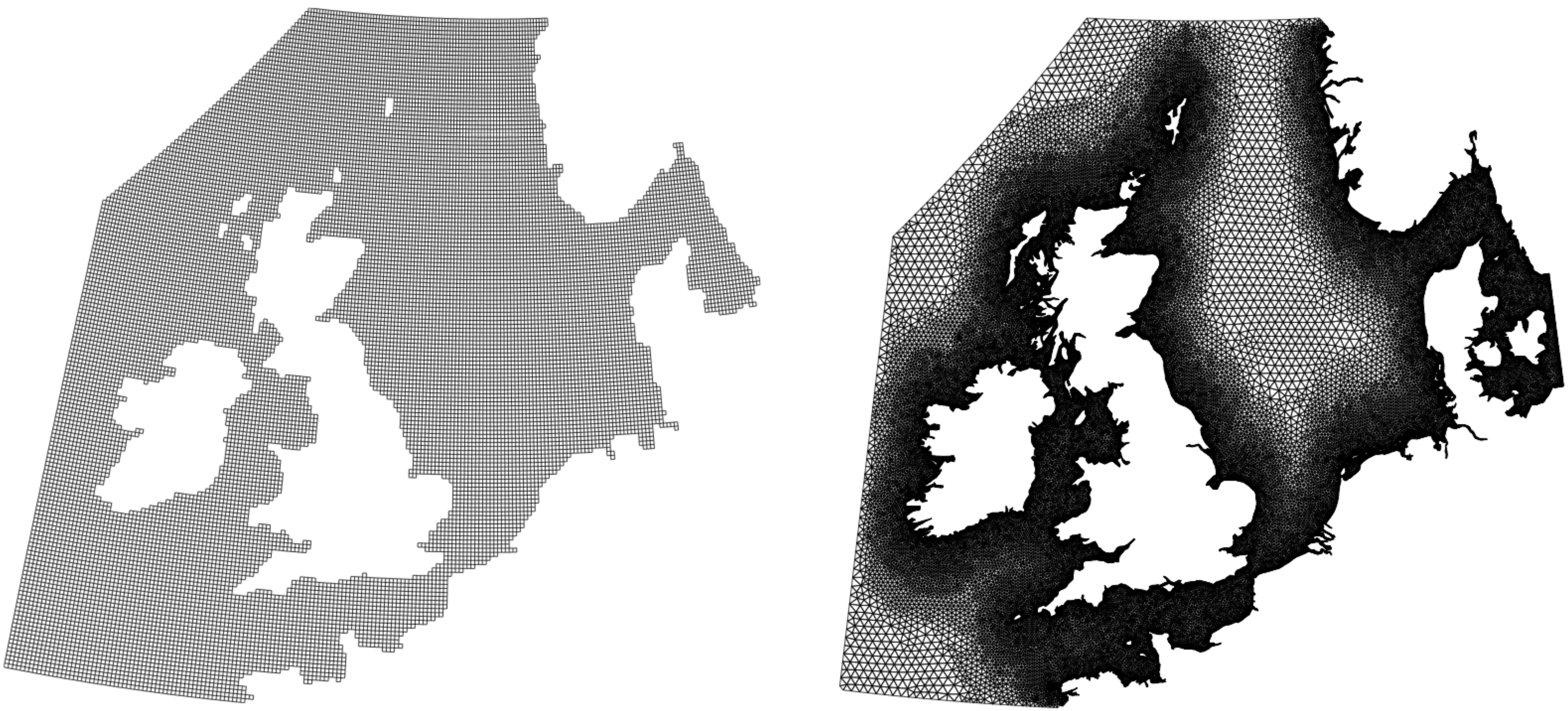
- Unstructured & structured meshes
- Hybrid: unstructured in the 2D + structured in the 3rd dimension = Extruded Meshes.





Advantages of Extruded Meshes of 2D unstructured base-meshes

Flexibility, Accuracy.



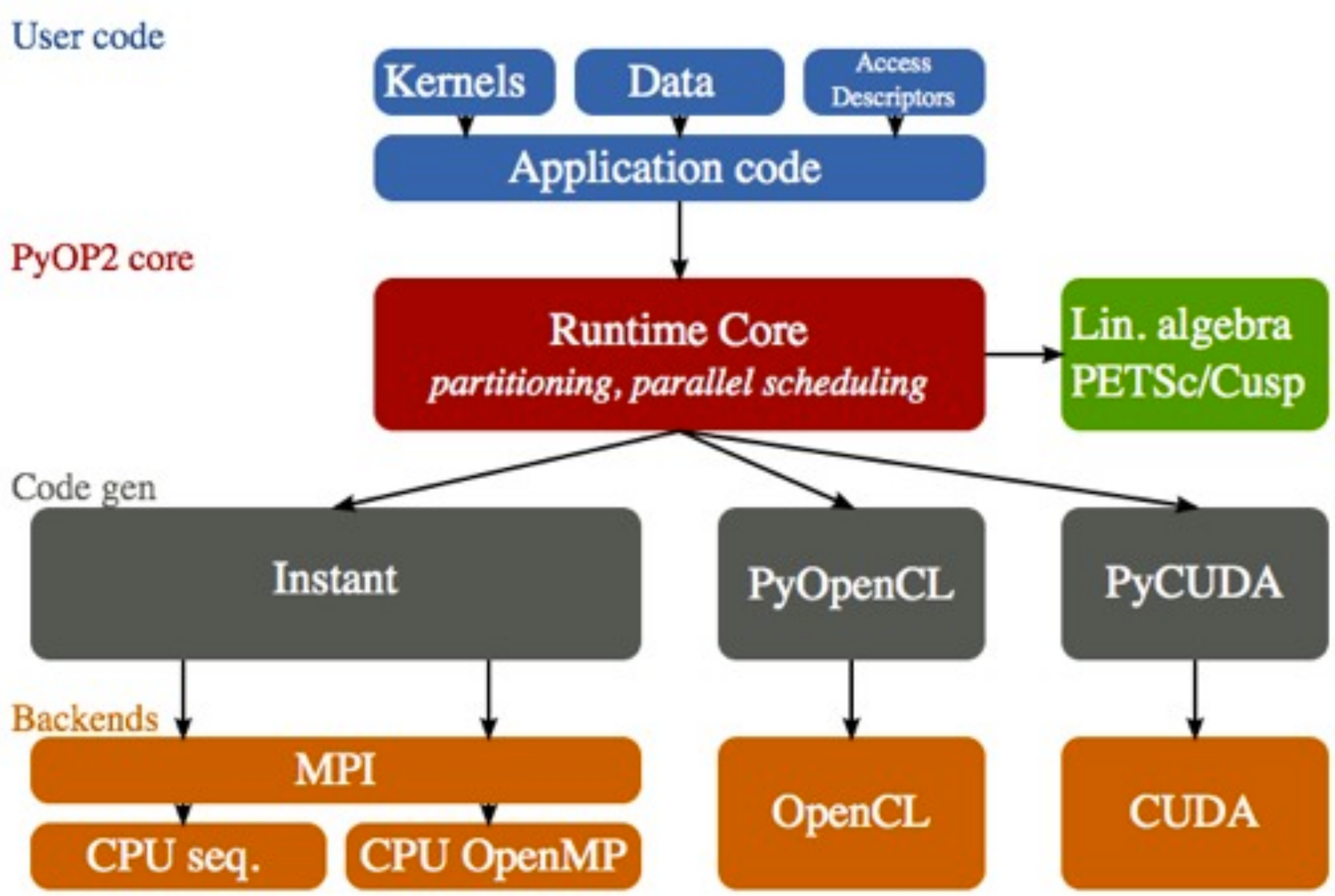
What do all these applications have in common?

The type of operations:

The application of the SAME computational kernel to EVERY member of a discrete set of mesh elements.

PyOP2

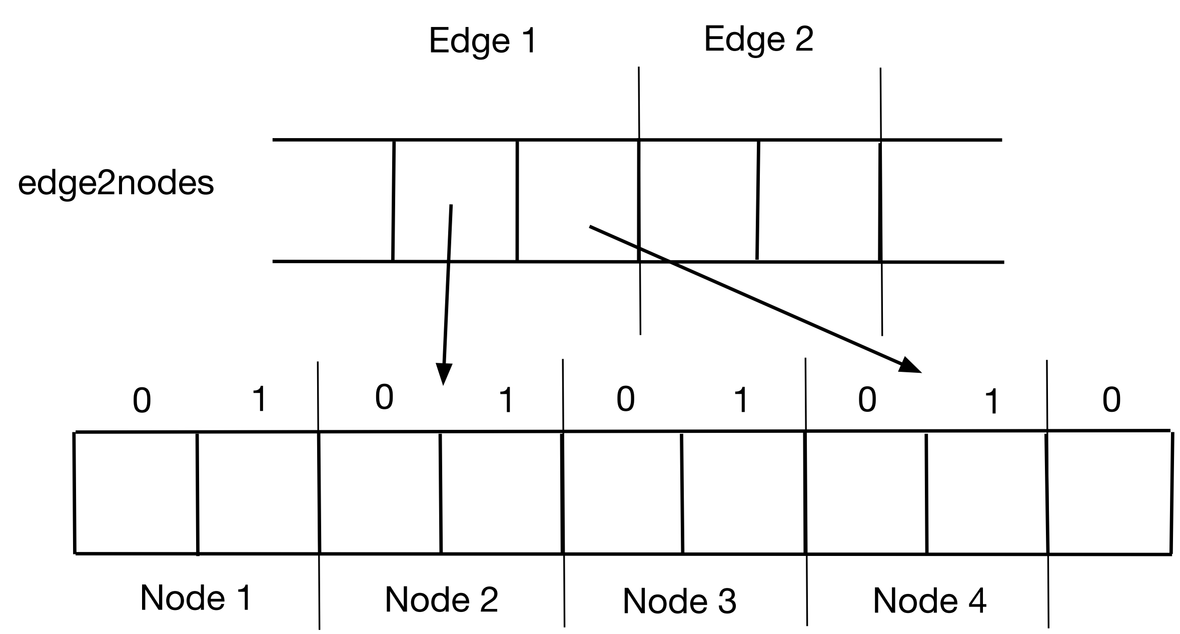
A Python implementation of the OP2 paradigm (Oxford Parallel Language for Unstructured Mesh Computations).



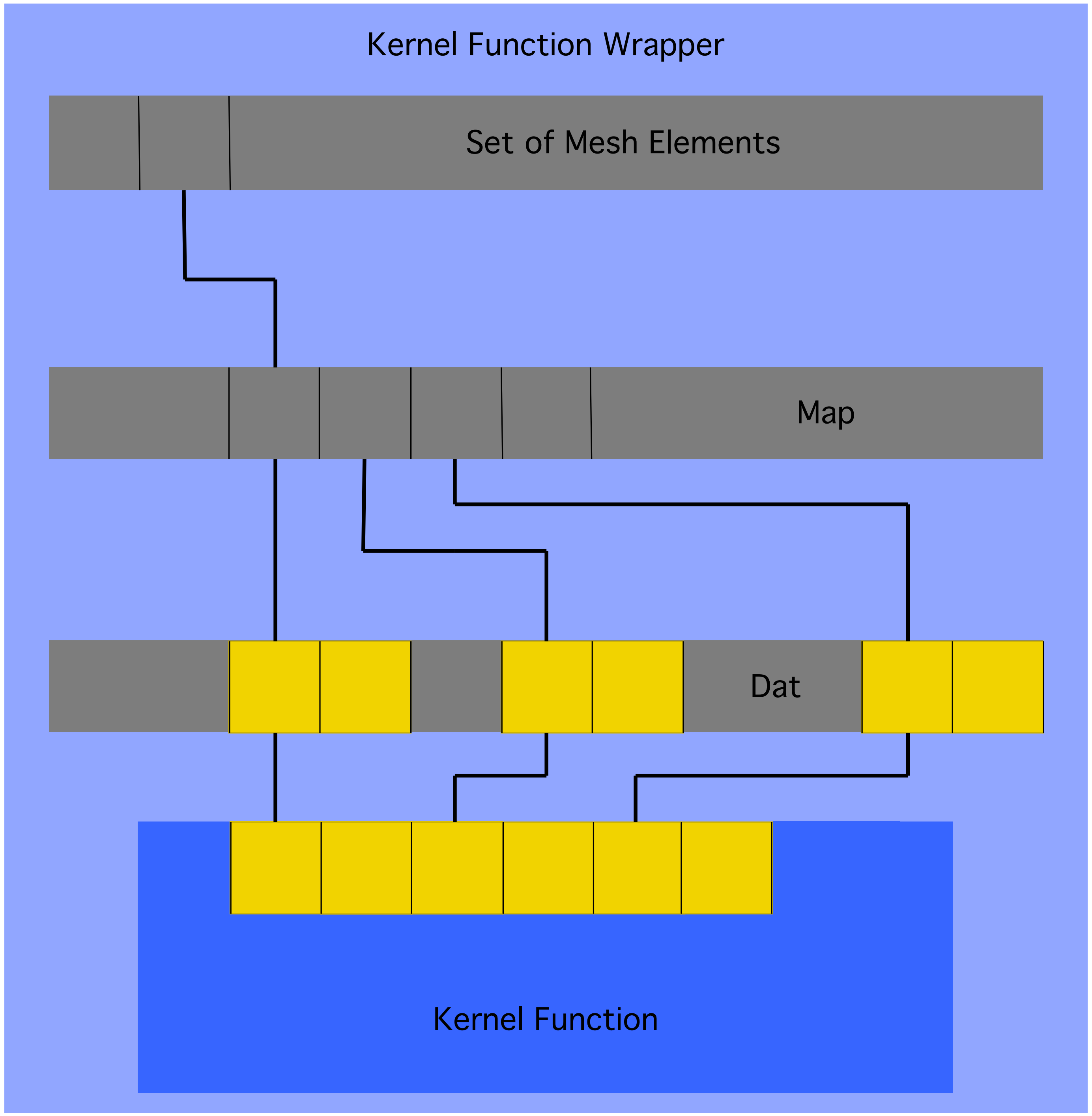
- Provides a high level Domain Specific Language (DSL) which translates code to a low level implementation through runtime code generation.
- Adds a new layer of abstraction for a flexible, portable and scalable implementation.

The PyOP2 DSL

- SETS for mesh elements;
- Data arrays (DATs) for fields, coordinates;
- MAPs for the connectivity of mesh elements;
- PARALLEL LOOPS for performing the actual work.



Code generation for indirect PyOP2 parallel loops

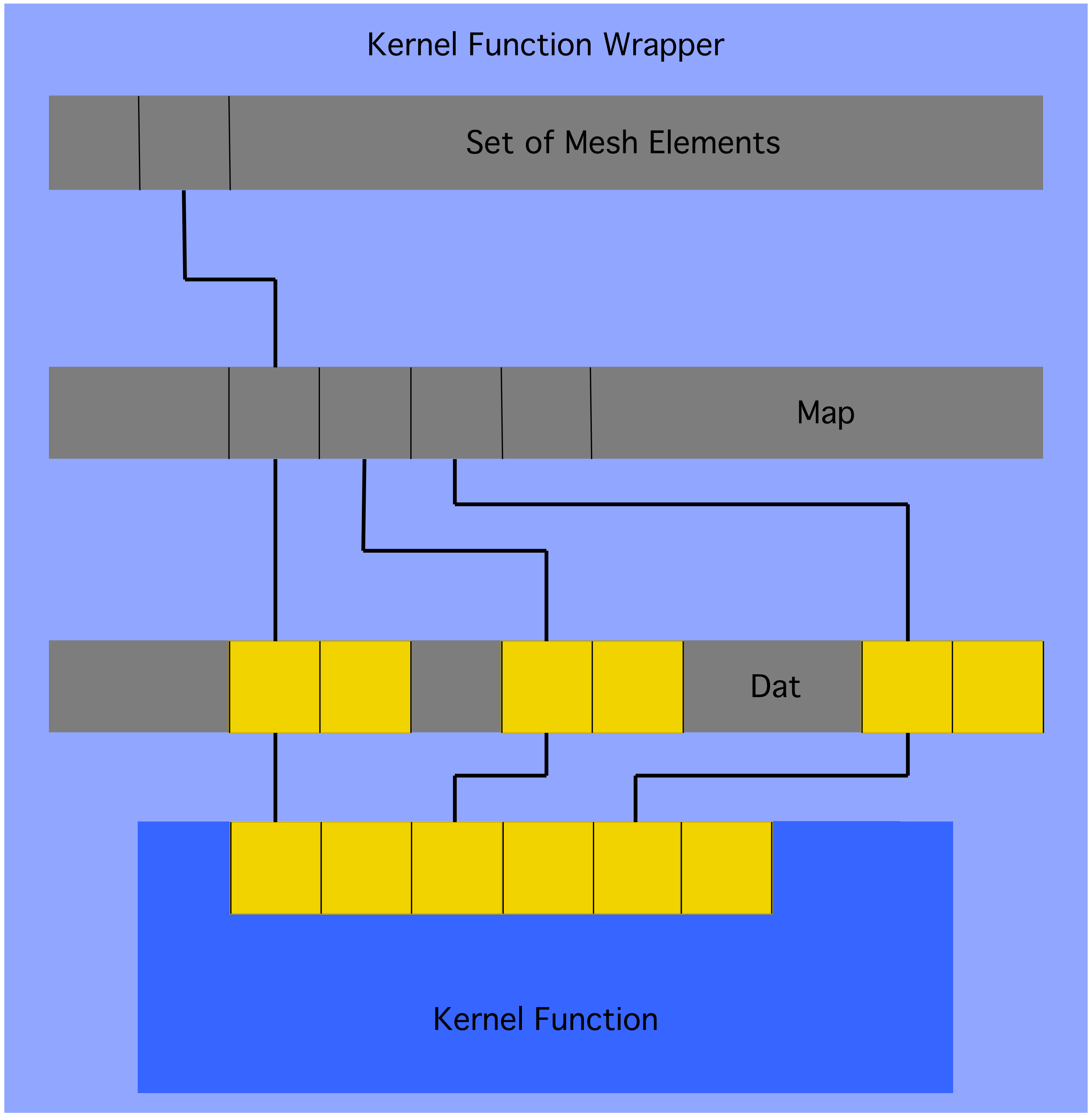


Iterate over mesh elements

For each element use the map to reference data.

Stage-in data to be used by the kernel.

Code generation for indirect PyOP2 parallel loops



Iterate over mesh elements

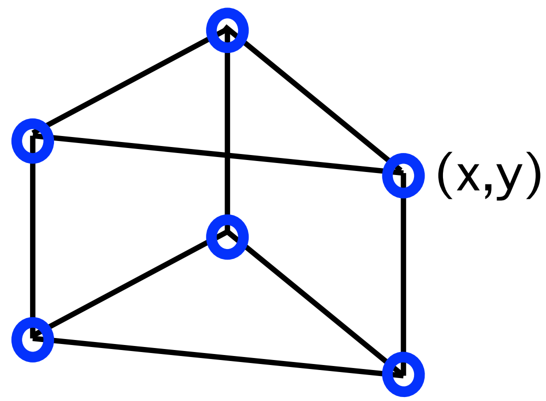
For each element use the map to reference data.

For each set of indirect element references iterate over the column elements.

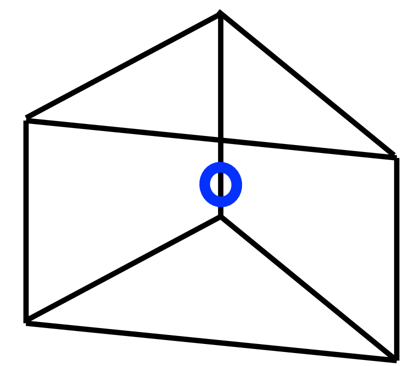
Stage-in data to be used by the kernel.

A Minimal Test Problem

$$\int T \mathrm{d}x$$



Coordinate Field: Location of Degrees of Freedom



Tracer: Location of Degrees of Freedom

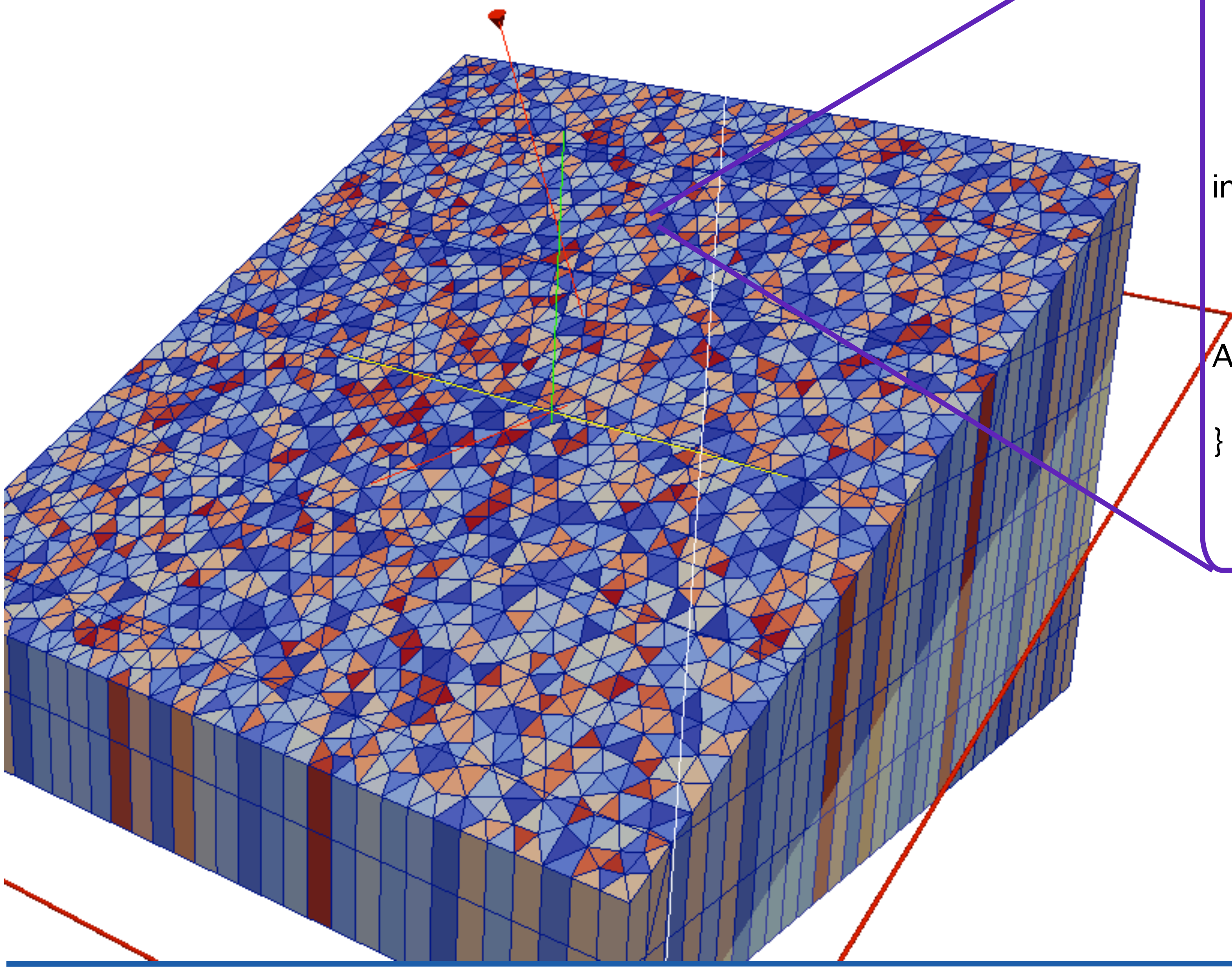
Effectively we are aiming to perform a very simple experiment: a global reduction operation.

No favours: The mesh we will be using is big enough to ensure that no cache benefits will be observed between time steps.

- The 2D unstructured mesh contains: 806,000 cells.
- There are 100 time steps executed in total.

Data movement dominates computation!

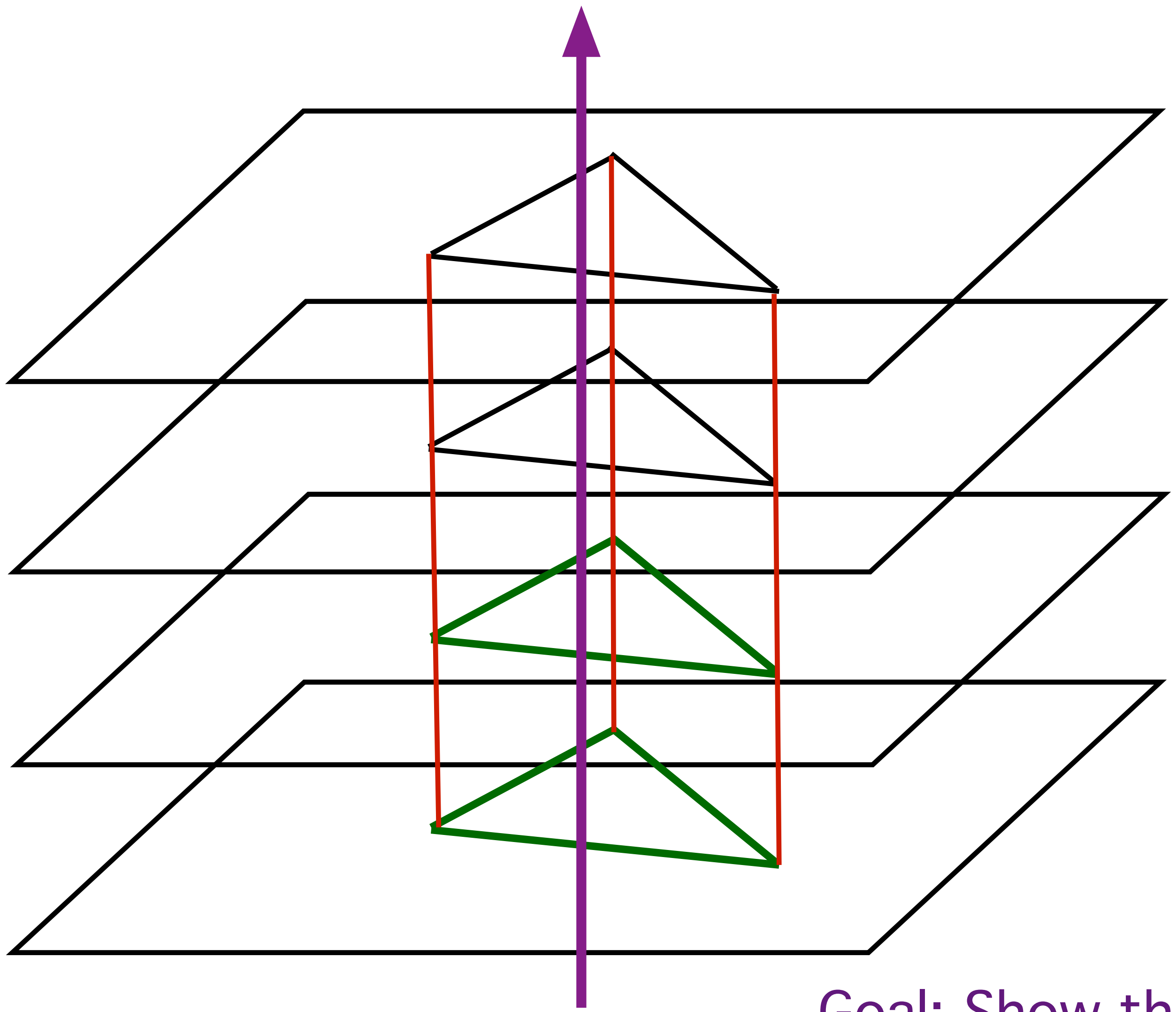
Kernel Application on extruded meshes



```
void comp_vol(double A[0],
              double *x[],
              double *y[],
              int j){

int area = x[0][0]*(x[2][1]-x[4][1]) +
           x[2][0]*(x[4][1]-x[0][1]) +
           x[4][0]*(x[0][1]-x[2][1]);

A[0] += 0.5*abs(area)*0.1*y[0][0];

}
```


Friday, 13 September 13

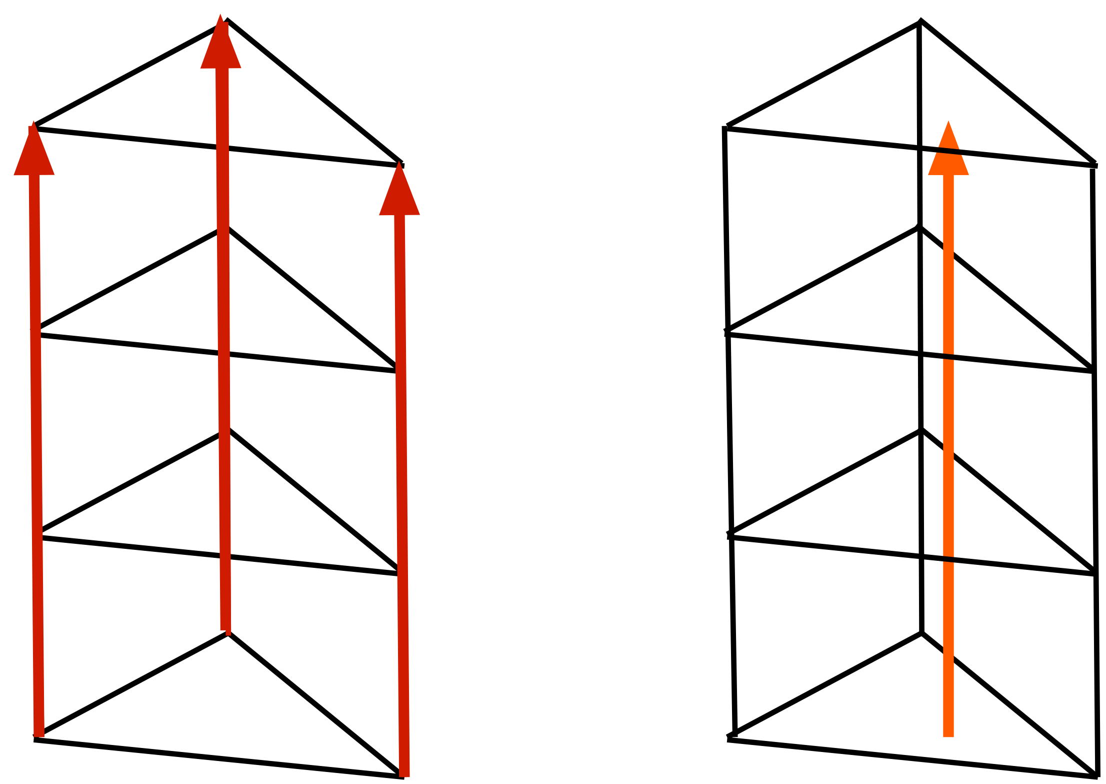
Using Extruded Meshes Efficiently



- We start from a 2D unstructured mesh.
- The 3rd dimension is structured.
- The innermost iteration occurs over the cells in the column.
- For each field we have just one indirection per column. Hence the penalty for the unstructured horizontal mesh is only paid once per column.

Goal: Show that the accesses in the structured direction remove the performance penalty of the unstructured direction.

Column Numbering - Vertical Data Locality

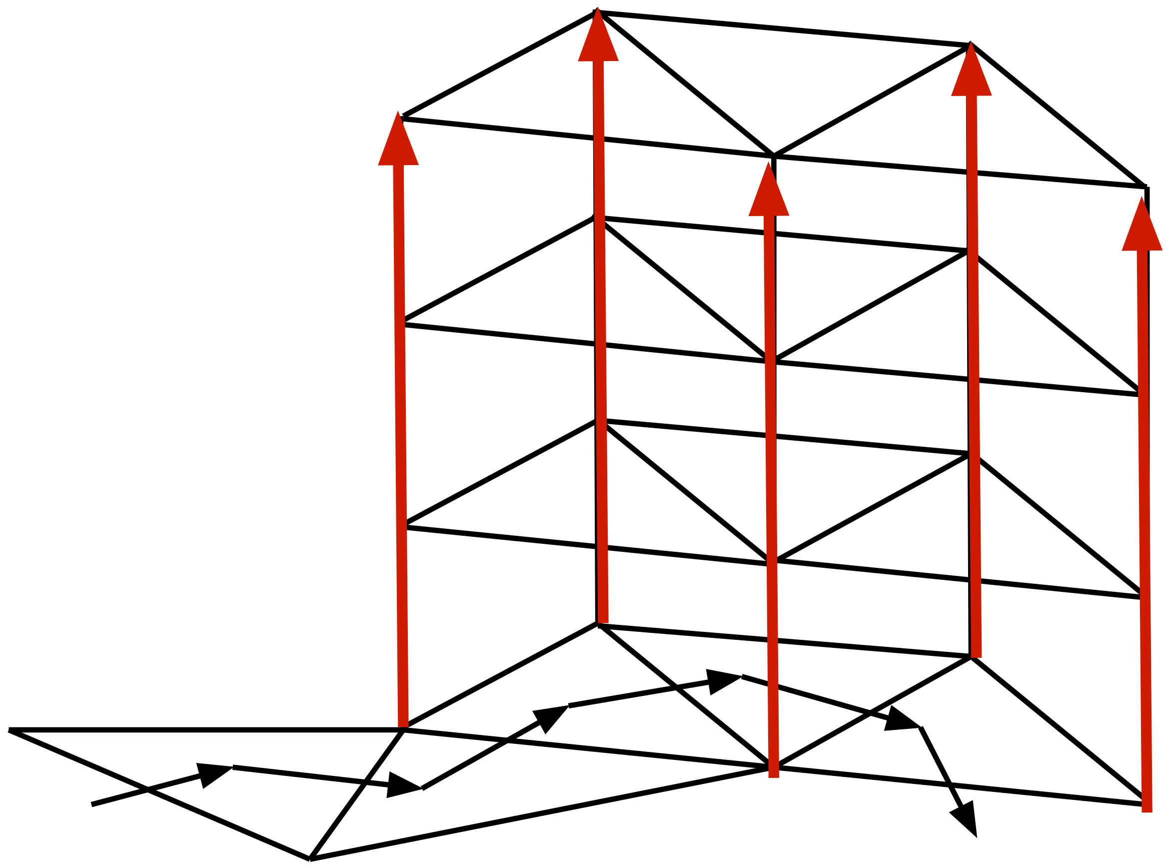


Vertical numbering of the mesh :

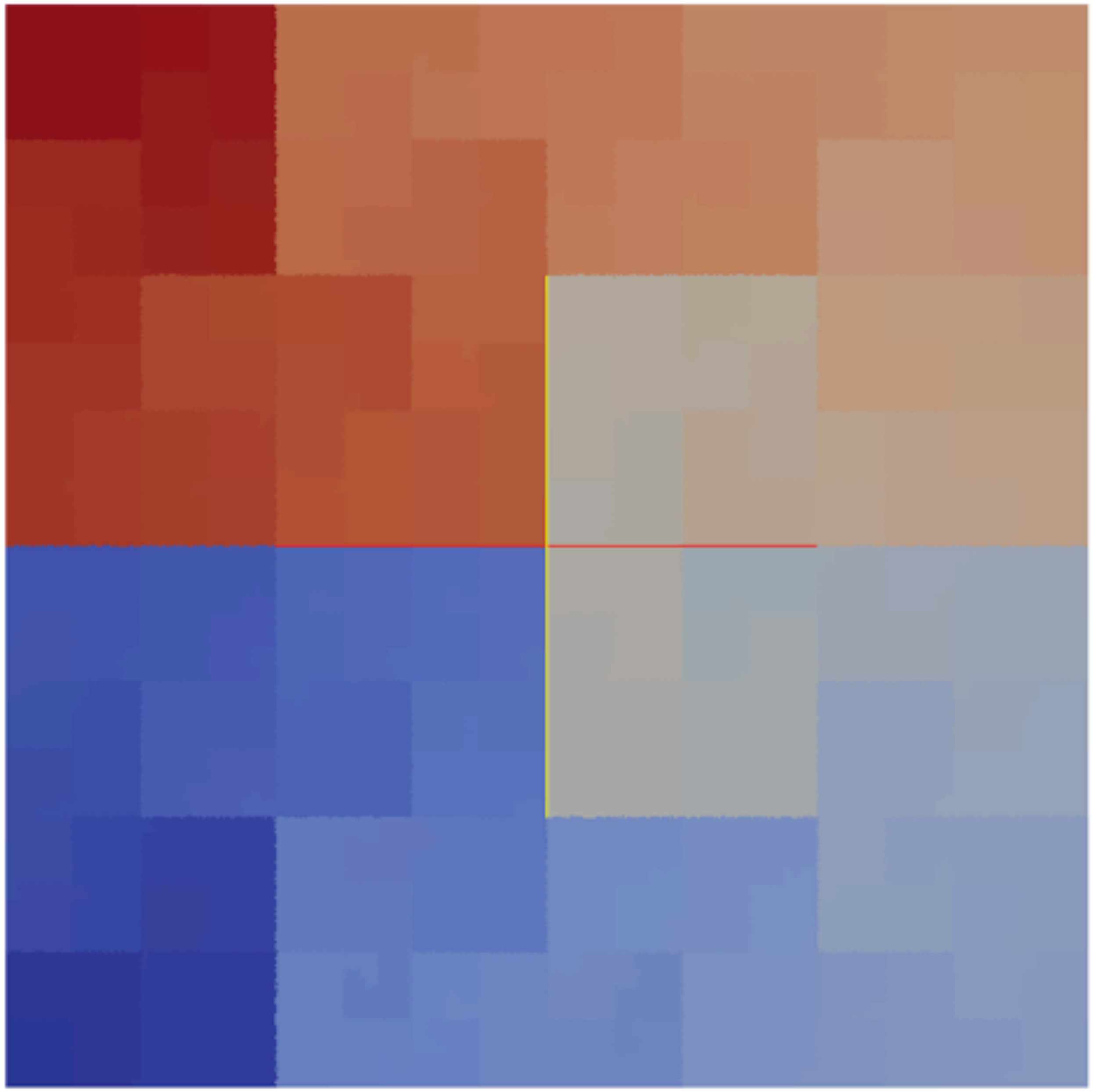
- Each group of degrees of freedom in the 2D will be “extruded” vertically for each of the layers.
- Numbering will be continuous as we want all the elements of the column to occupy a contiguous area in memory.

Mesh Numbering - Data Locality in the 2D

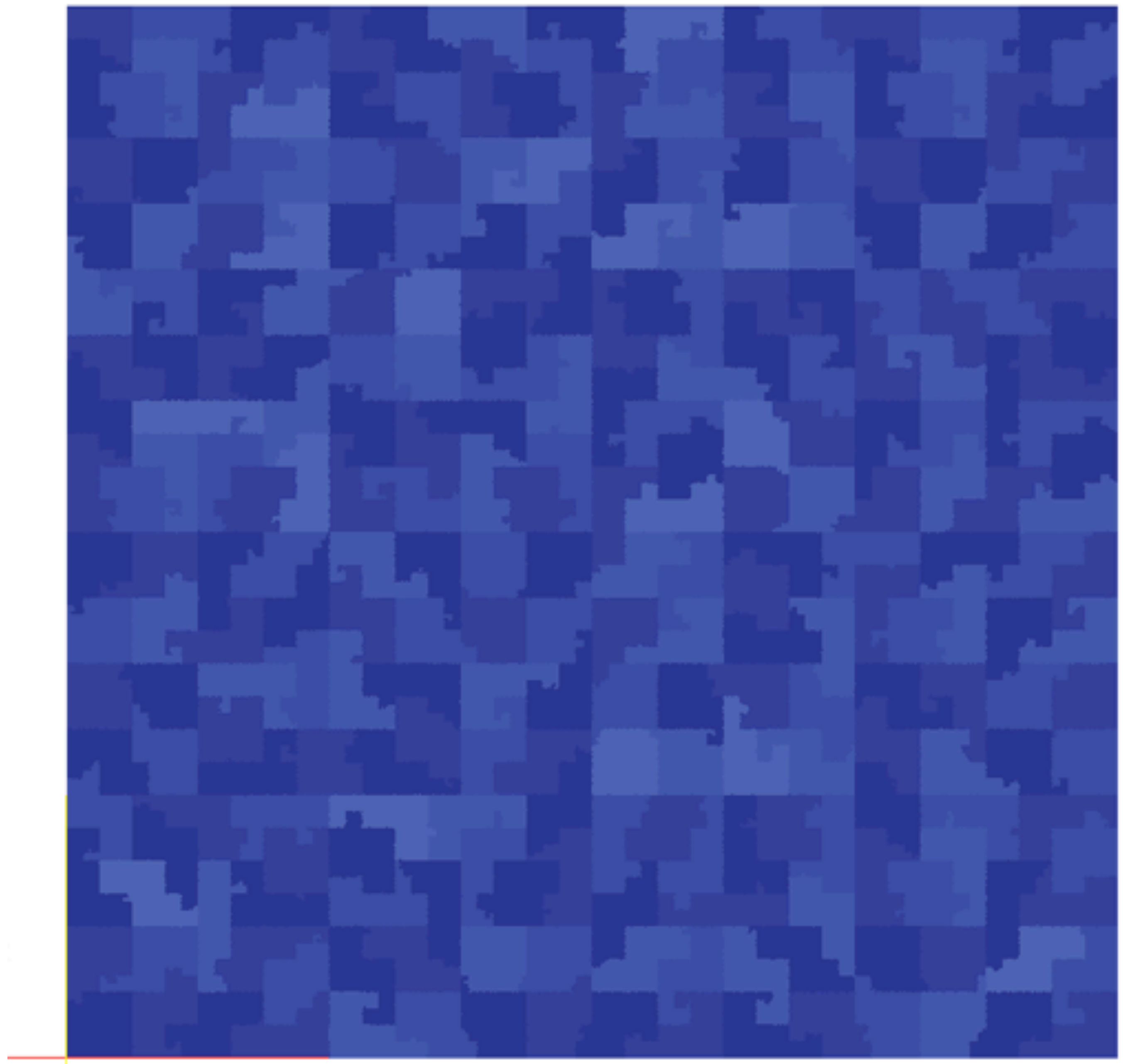
Using a space filling curve to renumber the 2D mesh will ensure temporal locality of the indirections.



This is how a good numbering looks:



Partitioning and Colouring

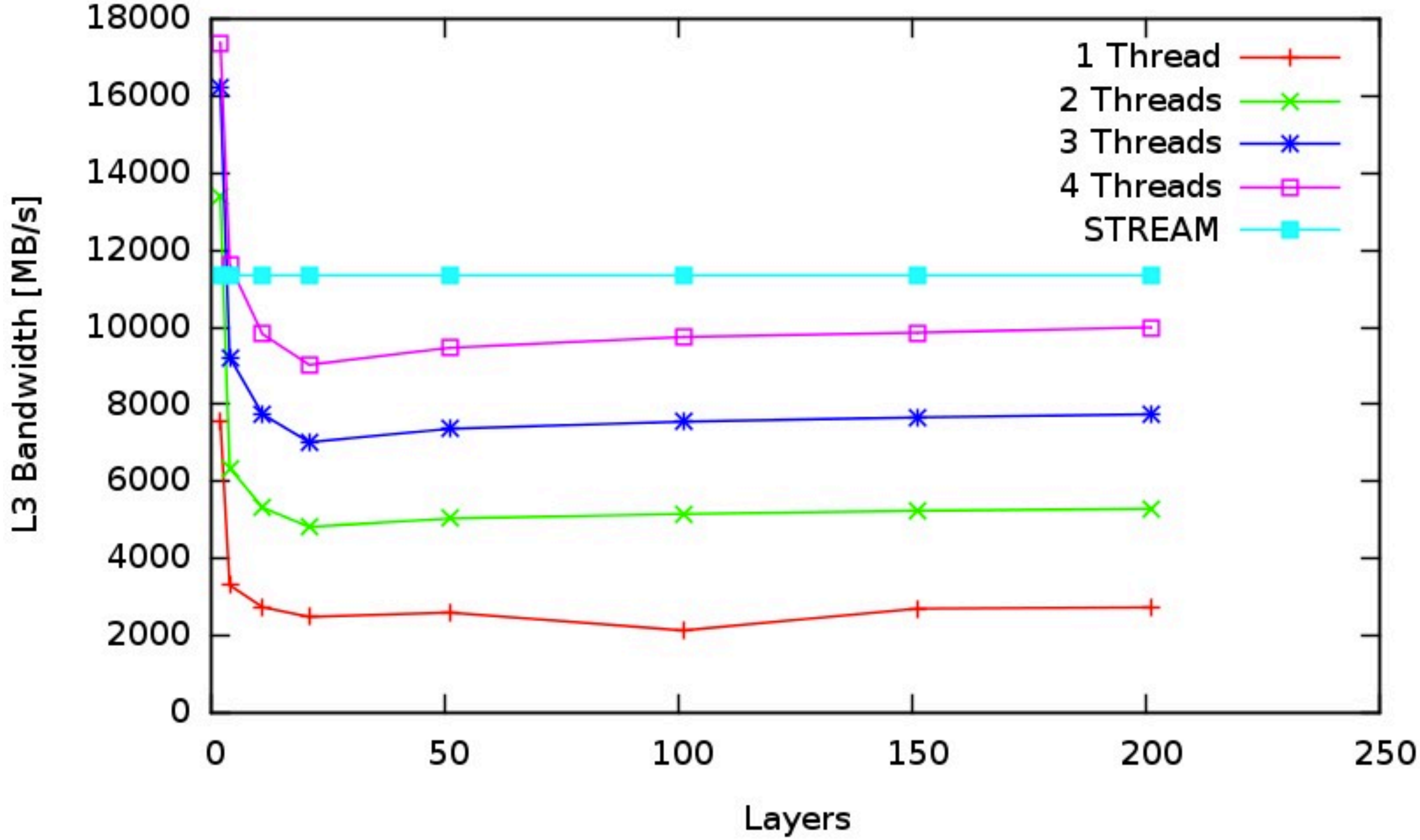


The hardware

```
CPU type:     Intel Core SandyBridge processor
*************************************************************
Hardware Thread Topology
*************************************************************
Sockets:        1
Cores per socket:     4
Threads per core:     2
-------------------------------------------------------------
Socket 0: ( 0 4 1 5 2 6 3 7 )
-------------------------------------------------------------

*************************************************************
Cache Topology
*************************************************************
Level:  1
Size:   32 kB
Cache groups: ( 0 4 ) ( 1 5 ) ( 2 6 ) ( 3 7 )
-------------------------------------------------------------
Level:  2
Size:   256 kB
Cache groups: ( 0 4 ) ( 1 5 ) ( 2 6 ) ( 3 7 )
-------------------------------------------------------------
Level:  3
Size:   8 MB
Cache groups: ( 0 4 1 5 2 6 3 7 )
```

- Intel 4-Core (SandyBridge) i7-2600 CPU @ 3.40GHz
- Memory topology diagram using Likwid.

L3 Cache Bandwidth STREAM Comparison using Likwid



Valuable Bandwidth

$$DV = Data\ Volume\ ;$$

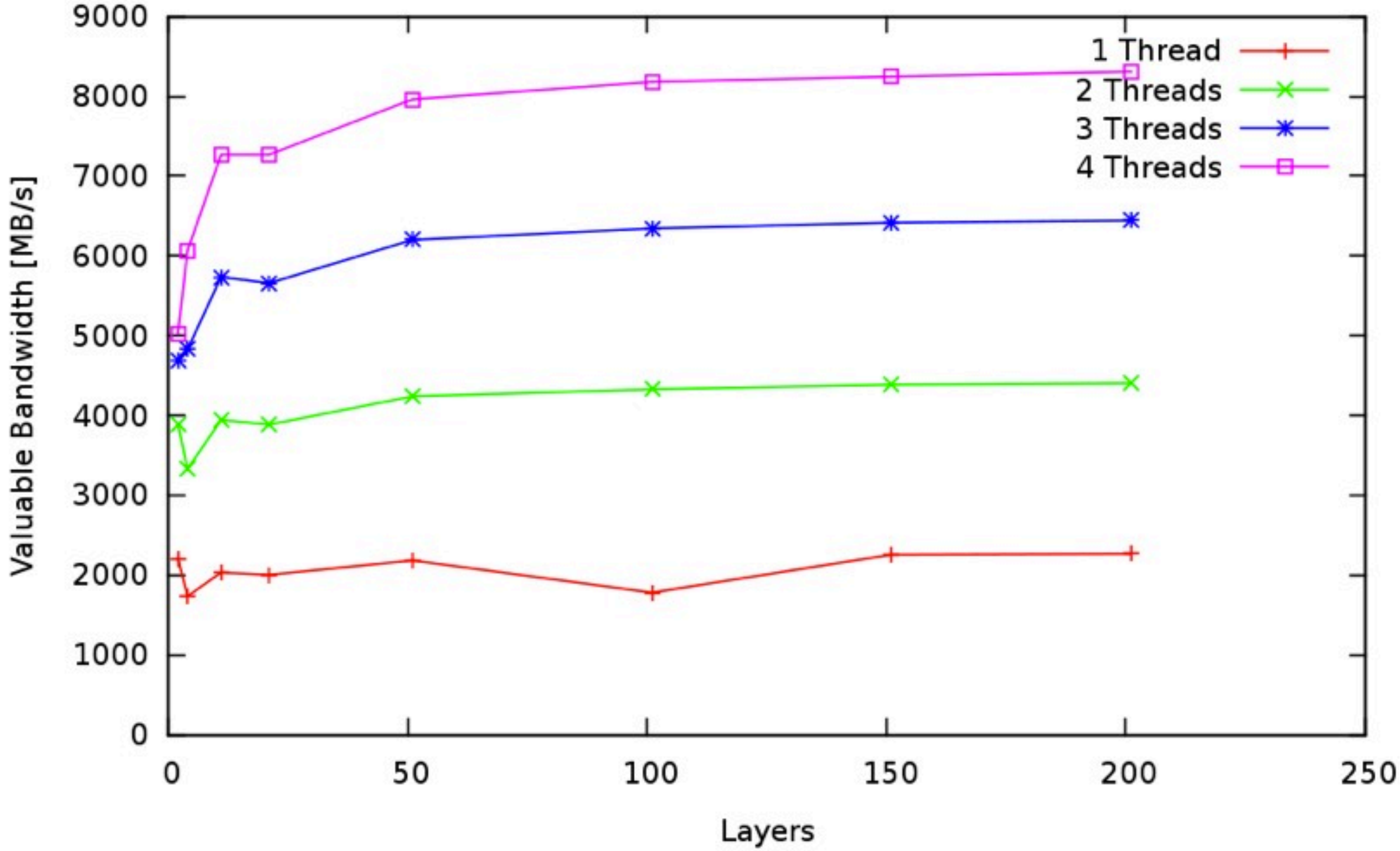
$$DV_{Coordinates} = Number\ of\ nodes \times Dimension \times Bytes\ per\ coordinate$$

$$DV_{Tracer} = Number\ of\ cells \times Bytes\ per\ tracer\ value$$

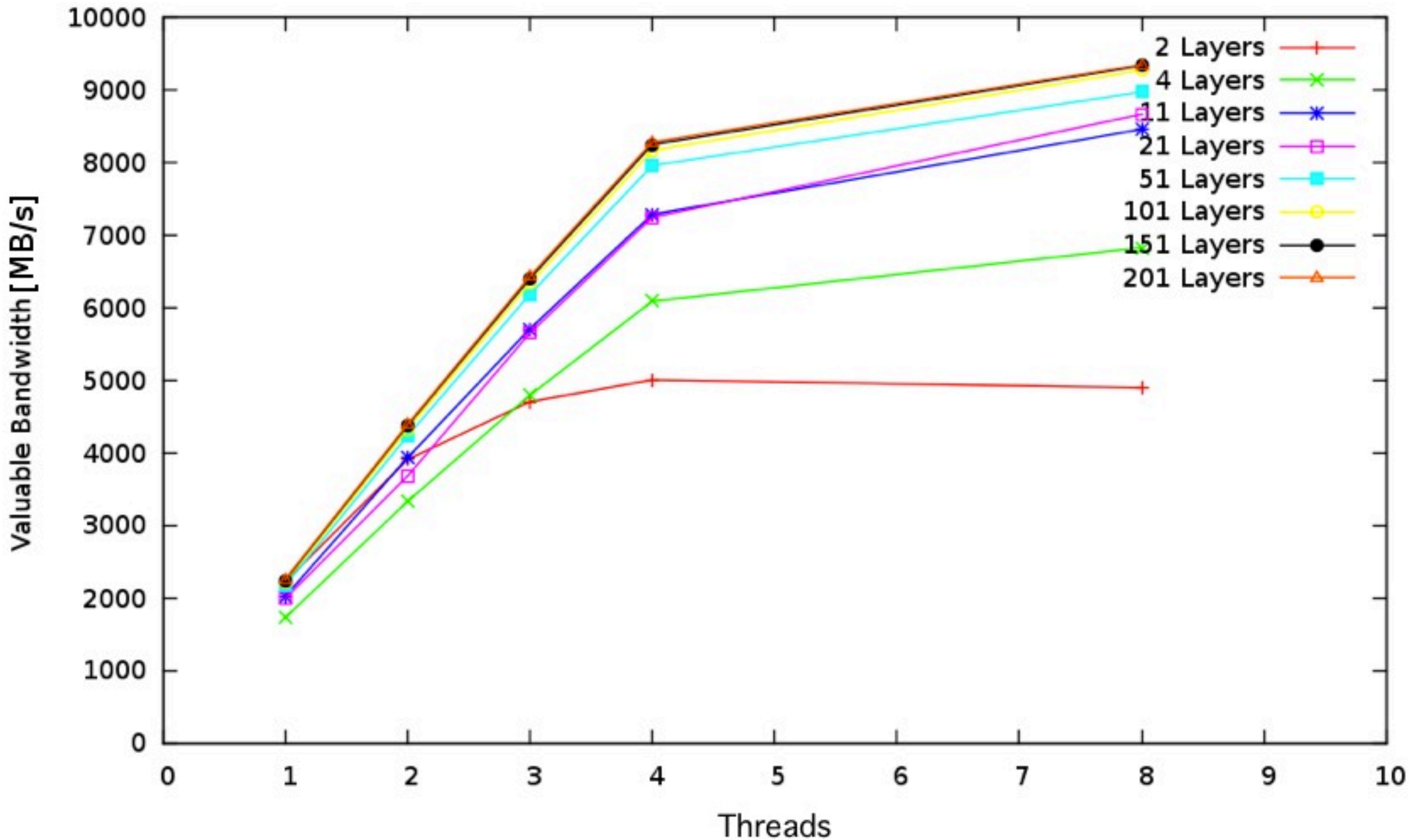
$$DV_{Total} = Outer\ Iterations \times Layers \times (DV_{Coordinates} + DV_{Tracer})$$

$$Valuable\ Bandwidth = \frac{DV_{Total}}{Execution\ time}$$

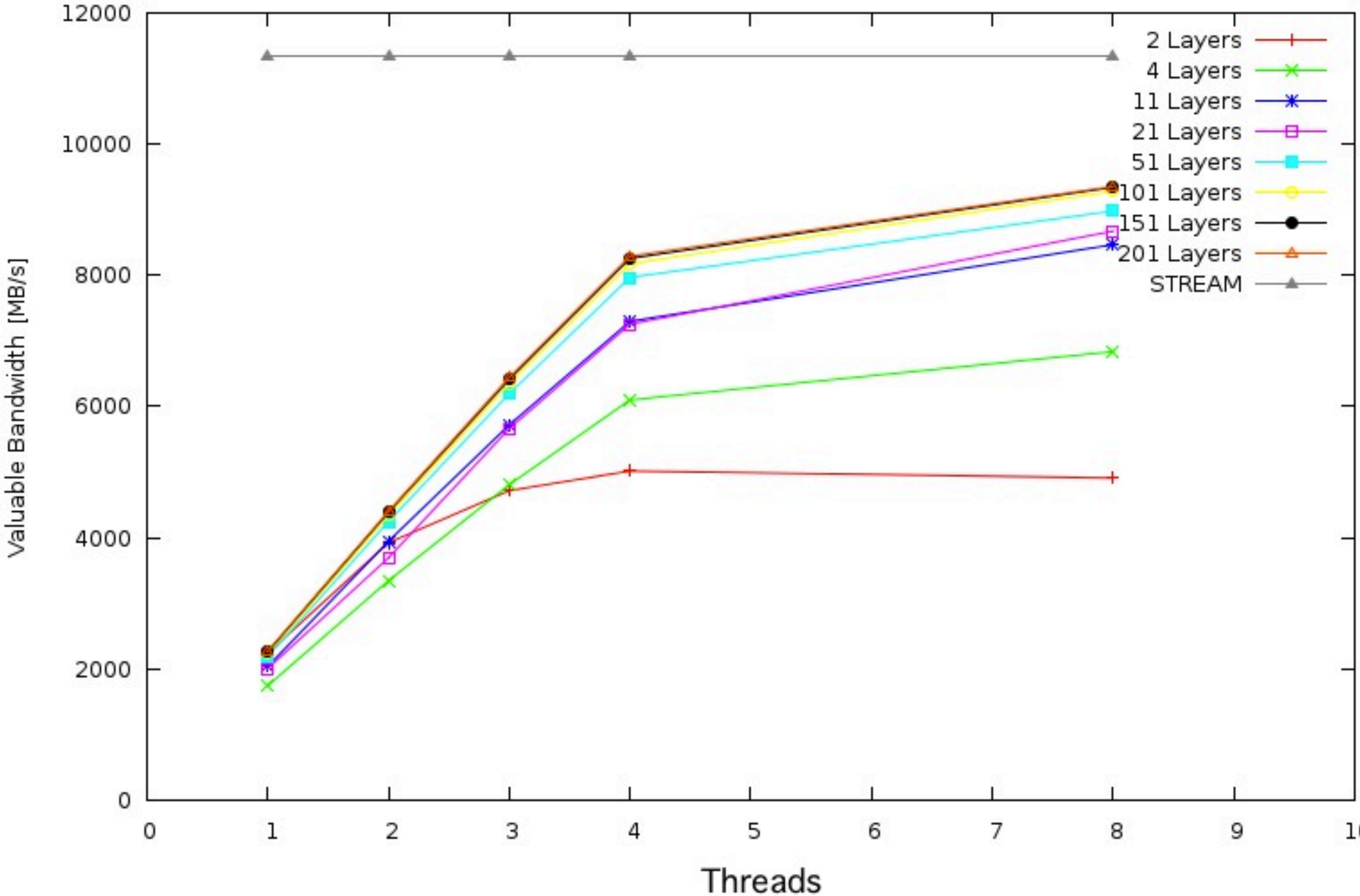
Valuable Bandwidth - a Lower Bound



Valuable Bandwidth - Increasing thread count



Valuable Bandwidth - STREAM Comparison



Conclusions for this experiment

We consider the Valuable Bandwidth achieved with 8 threads and more than 100 layers and compare it with the STREAM bandwidth.

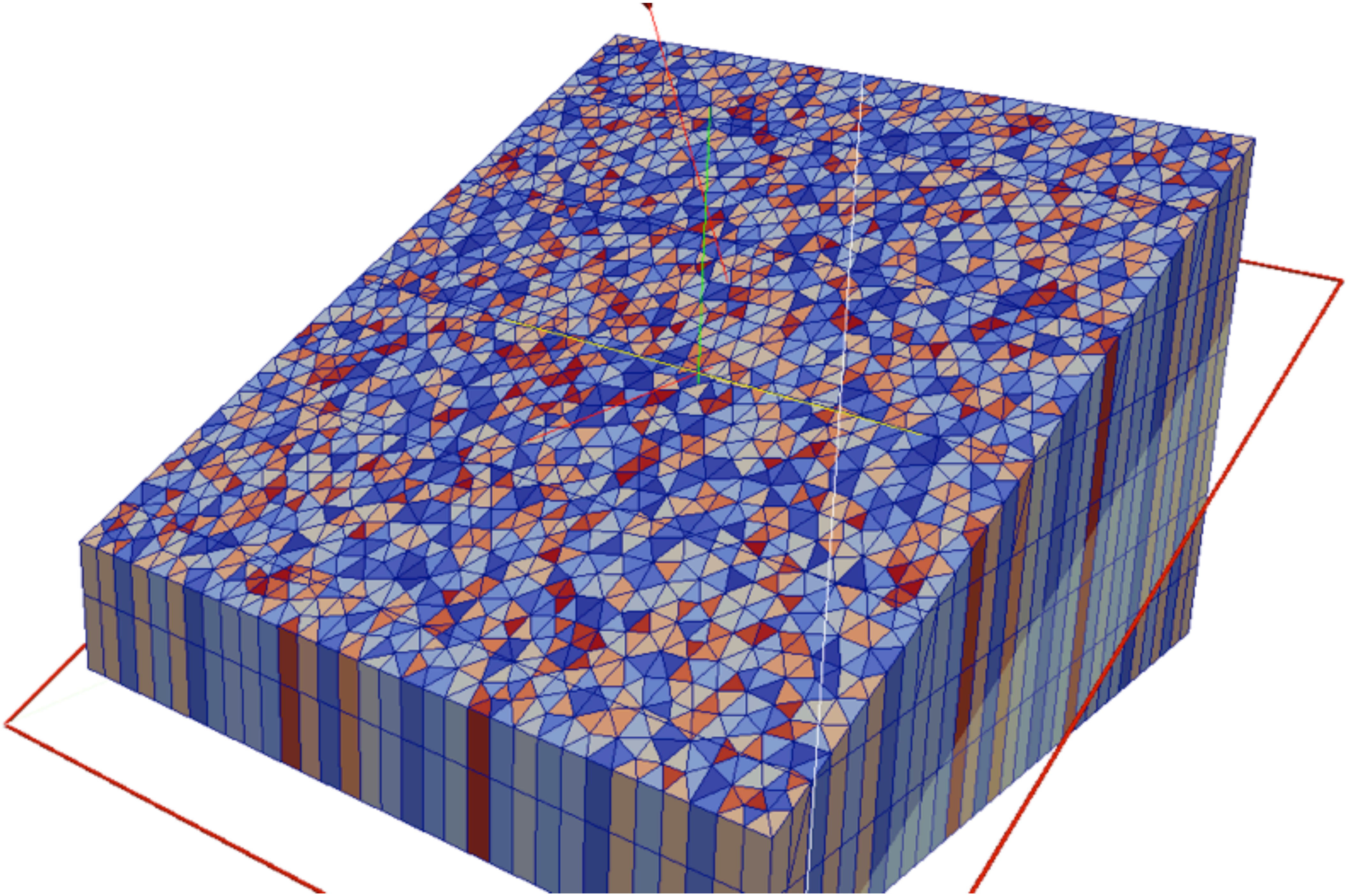
The Valuable Bandwidth achievement of this bandwidth stress test is 82.4% of the STREAM benchmark bandwidth.

The number of layers needed to offset the penalty of using an unstructured mesh is about 20.

Remarks

- We now know what makes a good Extruded Mesh.
- Location, location, location!
- Comparison with STREAM rather than a Structured Mesh code.
- Different slices through the memory hierarchy performed with Likwid show similar performance numbers to the STREAM benchmark.
- Limitations: only reading, only one platform, only single socket.

Thank you!

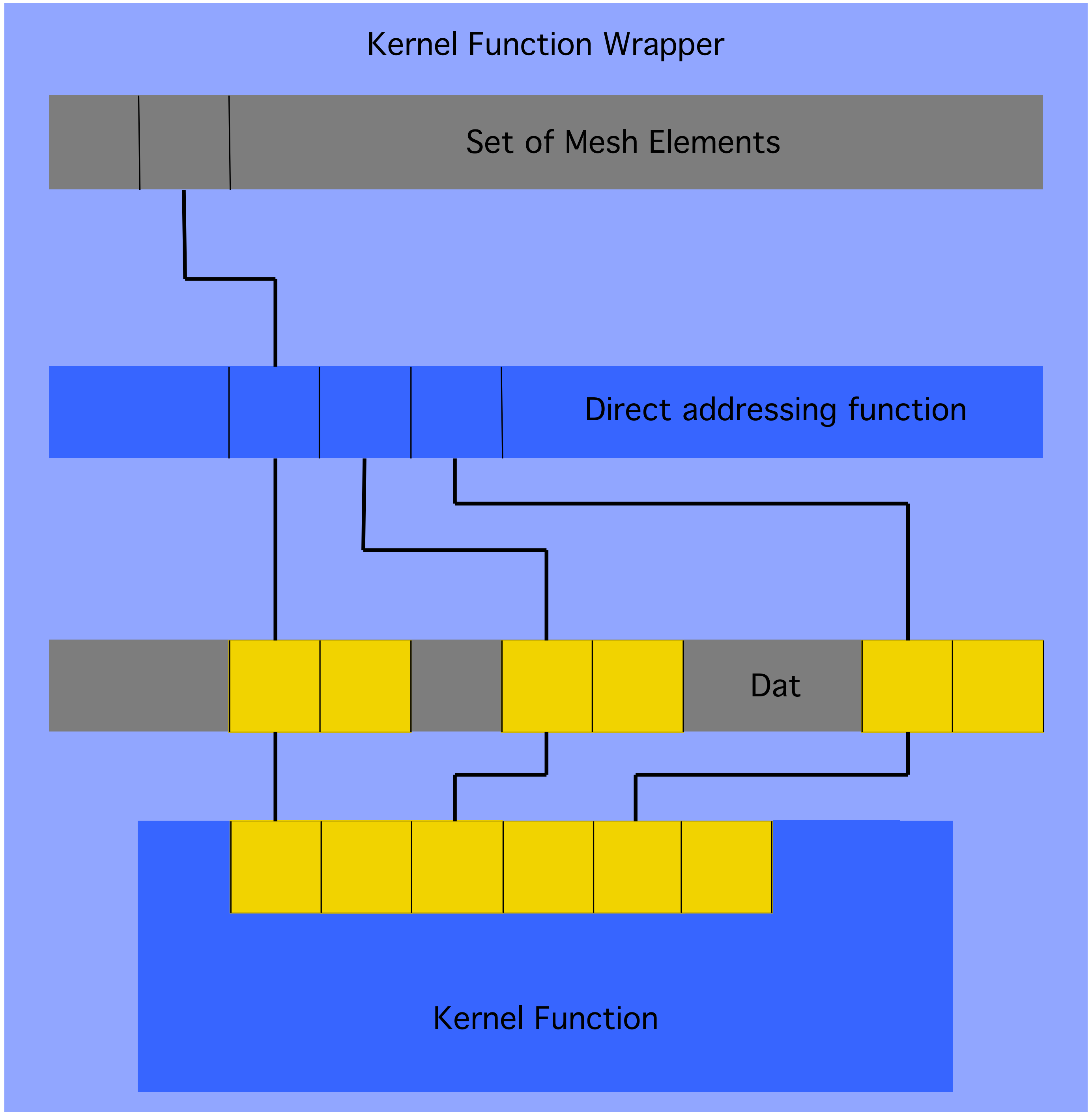


Solving Partial Differential Equations

```
# Define mesh, function space
mesh = Mesh("box_with_dent.xml.gz")
V = FunctionSpace(mesh, "CG", 1)

# Define basis and bilinear form
u = TrialFunction(V)
v = TestFunction(V)
a = dot(grad(u), grad(v))*dx

# Assemble stiffness form
A = PETScMatrix()
assemble(a, tensor=A)

# Create eigensolver
eigensolver = SLEPcEigenSolver(A)

# Compute all eigenvalues of A x = \lambda x
print "Computing eigenvalues. This can take a minute."
eigensolver.solve()
```

- Means starting from a high level specification of the problem and ending up with a low-level optimised implementation.
- The FEniCS - Dolfin tool chain already does something similar:
    - Uses the Unified Form Language (UFL) to specify the problem.
    - Uses the FEniCS Form Compiler (FFC) to automatically generate the kernel code.
    - Uses the Dolfin backend to provide the code required to run the kernel function.

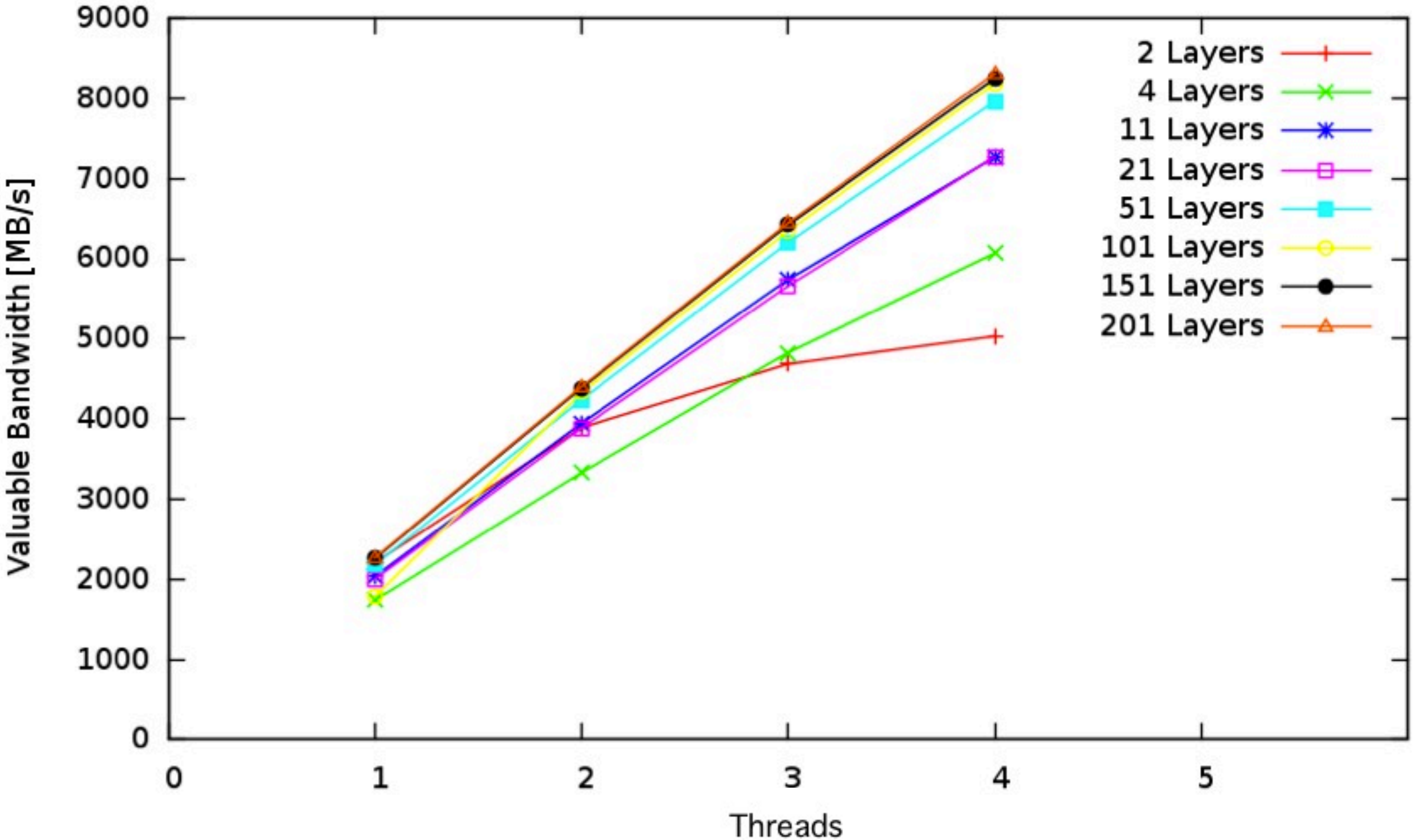
A PyOP2 parallel loop - direct



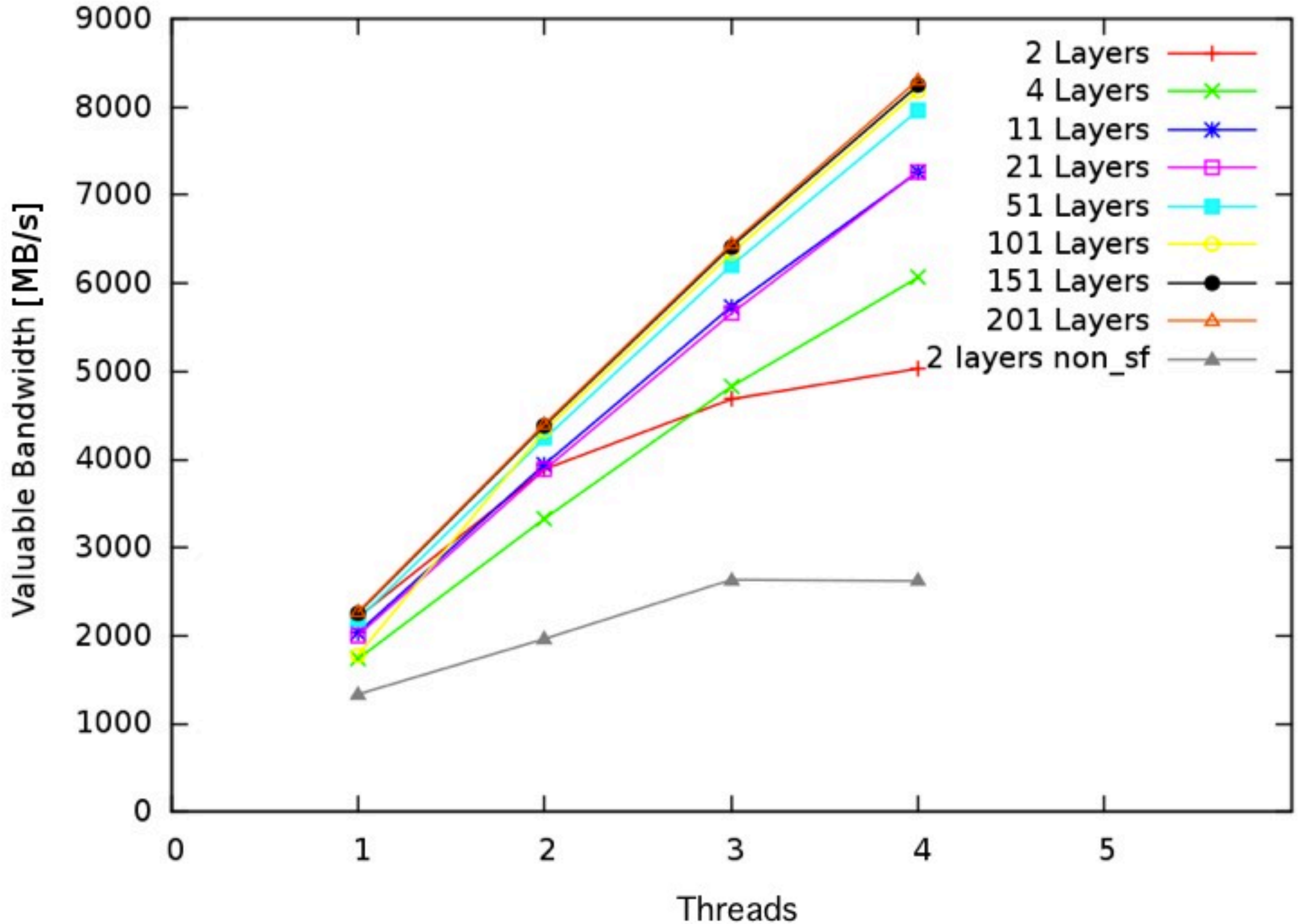
Considerations for Exploiting the Structure of Data

- There is a tight coupling between the structure of the mesh and the structure of the data.
- Performance is affected as the problem structure has a direct impact on data movement.
- Moving data efficiently leads to improved scalability - saturating the bandwidth is not a question of “if” but a question of “when”.
- Exploiting structure requires detailed knowledge of the particularities of each system architecture - different micro-optimisations are required for different architectures so this affects portability.
- Being able to seamlessly switch between implementations provides flexibility.

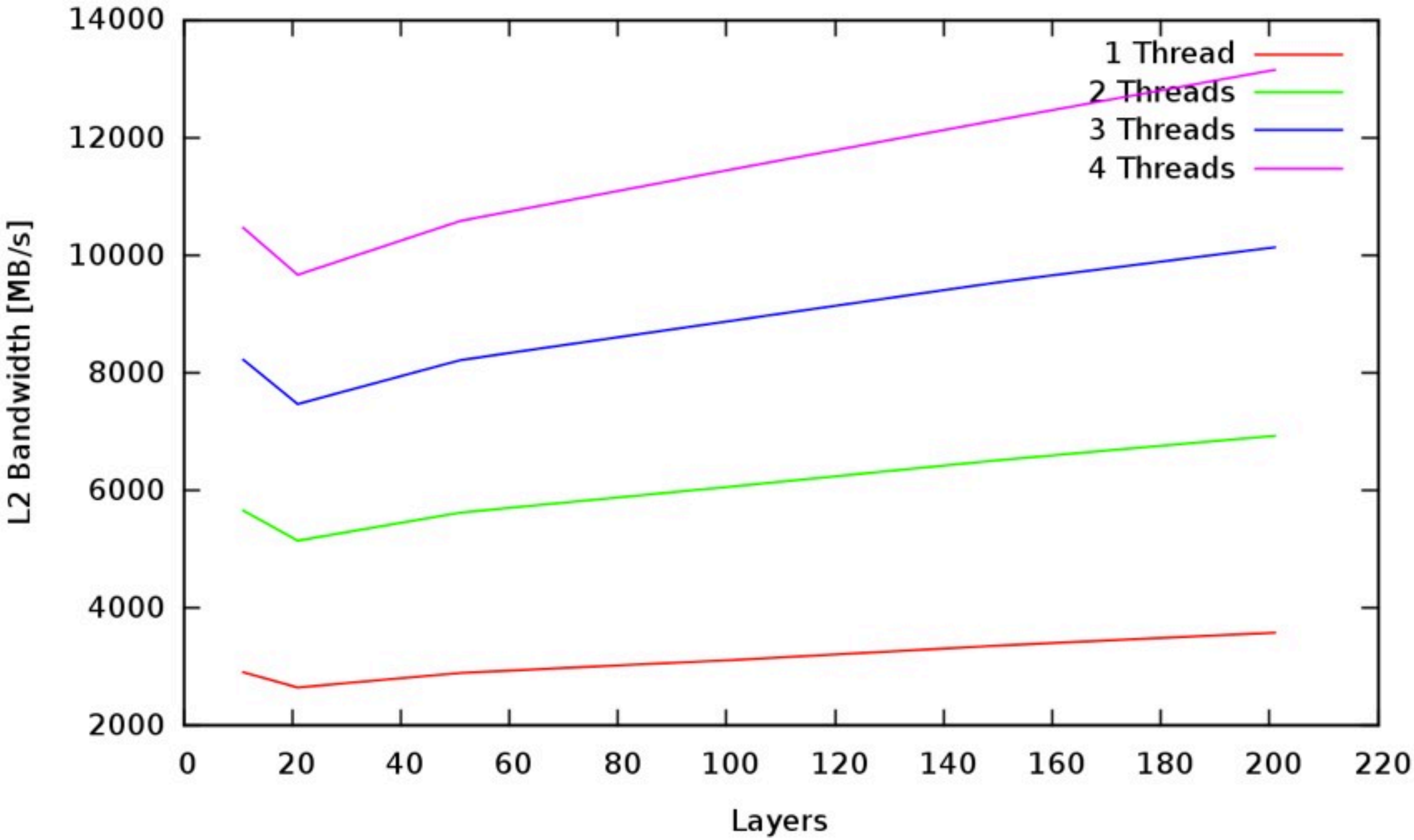
Valuable Bandwidth - a Lower Bound



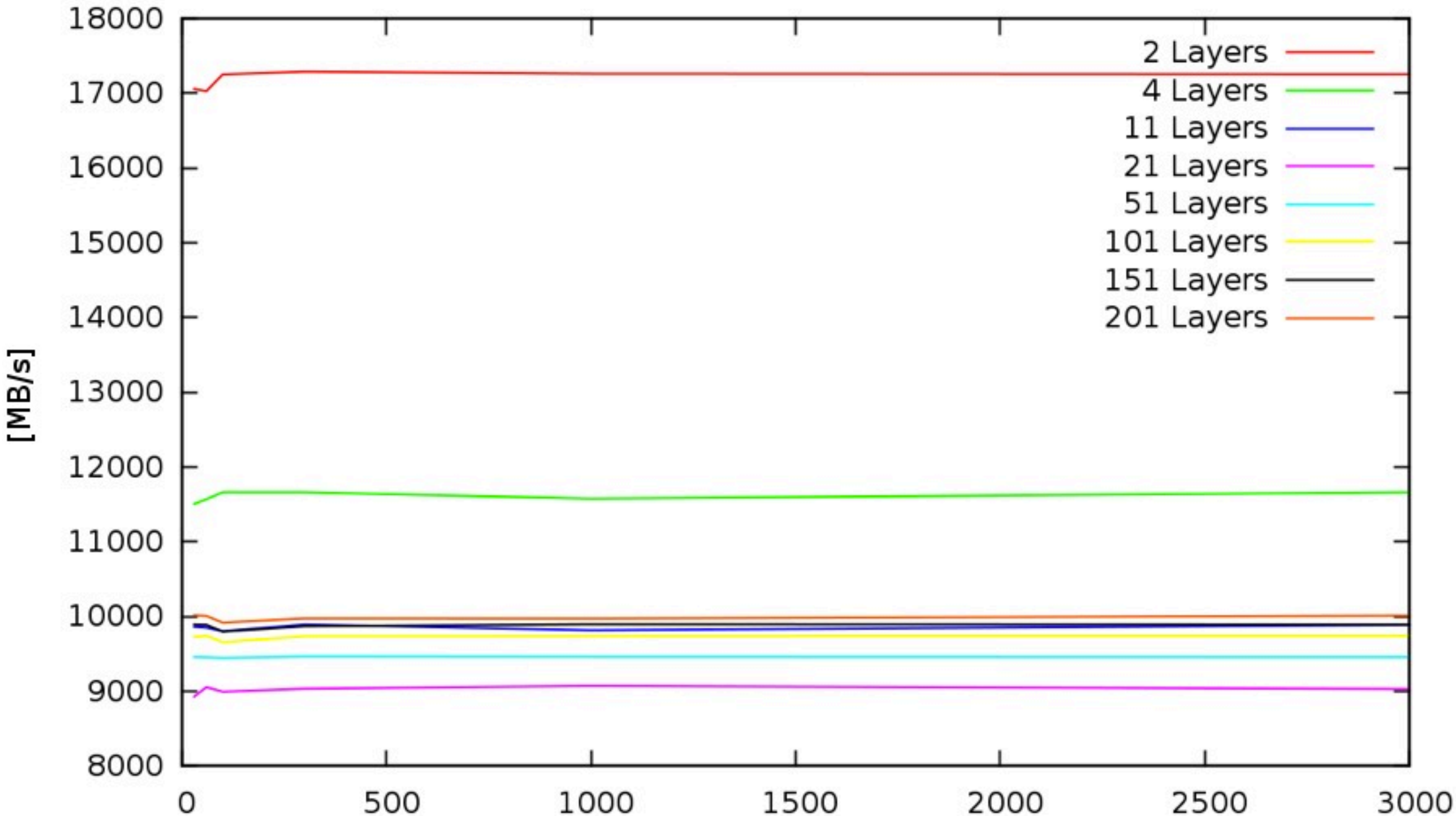
Valuable Bandwidth - a Lower Bound



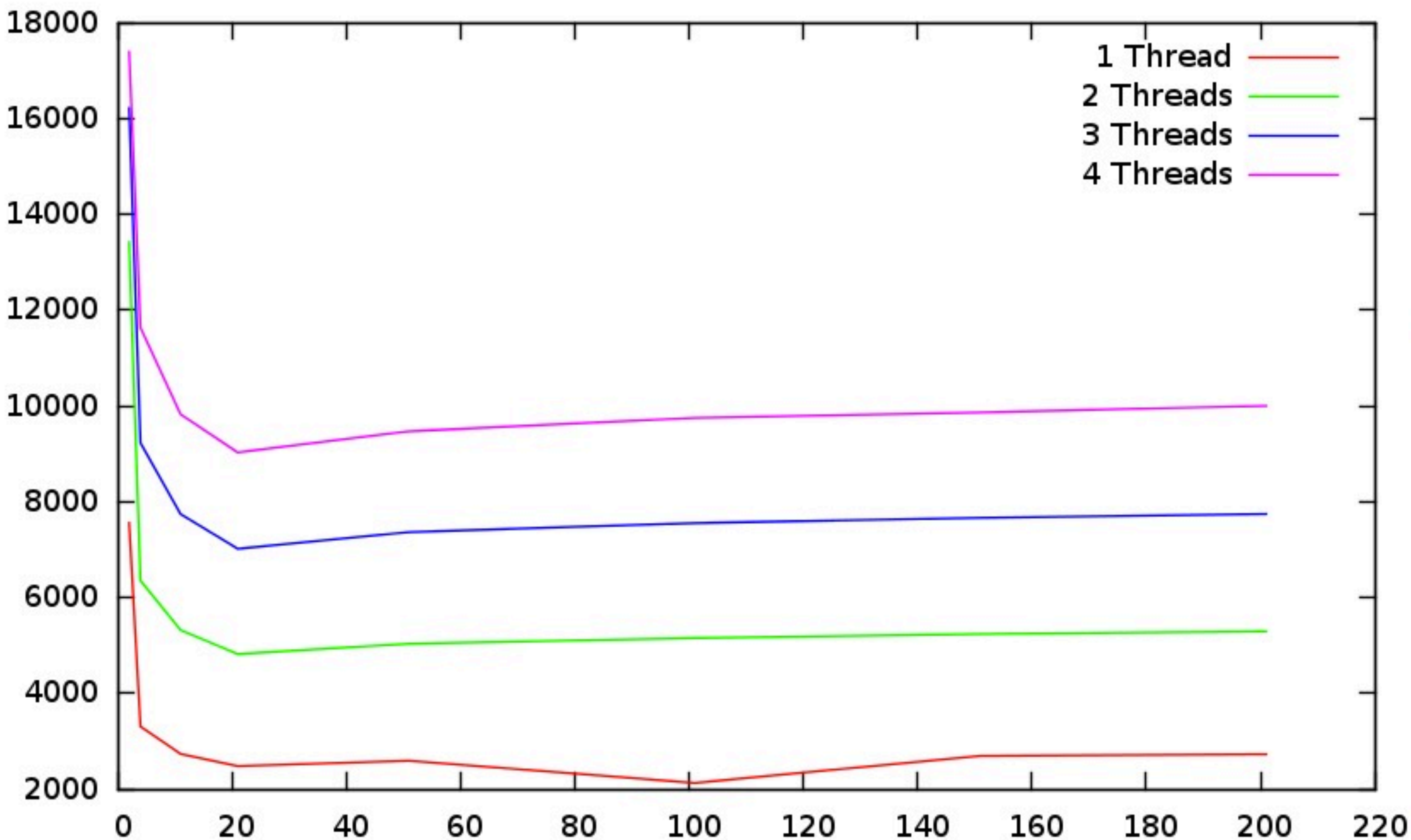
L2 Cache Bandwidth using Likwid



Partition Independence



L3 Bandwidth (Likwid) - Layers vs. Threads



Iterating over the Mesh

- for each colour C
  - for each partition P in C
    - for each 2D cell in partition P
      - for each cell in the column
        - apply Kernel