Optimizing Performance of the Lattice Boltzmann Method for Complex Structures

Stefan Donath

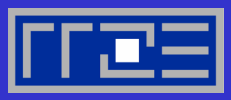
Thomas Zeiser, Gerhard Wellein, Georg Hager, Johannes Habich

Friedrich-Alexander University Erlangen/Nuremberg

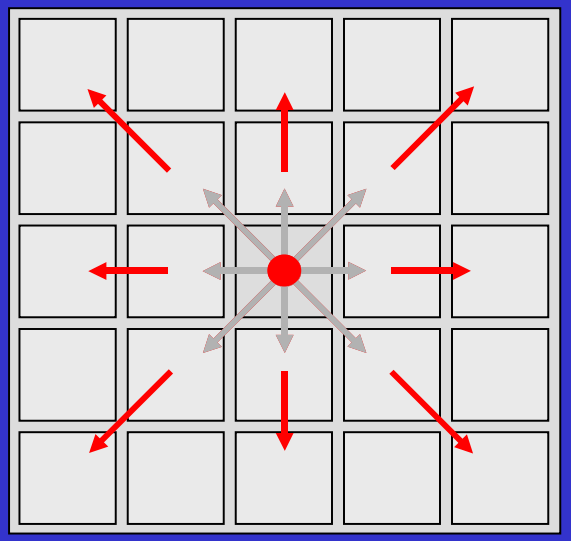
Regional Computing Center of Erlangen (RRZE)

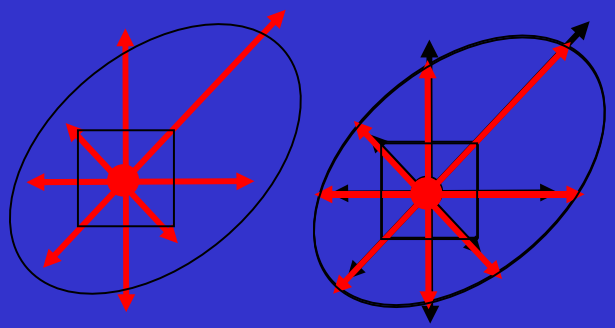
Outline

- Introduction
  - Lattice Boltzmann Method
  - Implementation Aspects
  - Application Example
  - Introduction of Complex Geometries used
- Implementation
  - 1D-Array Implementation
- Optimization
  - Space-Filling Curves
  - Blocking
  - Memory Layouts
- Results
- Conclusion

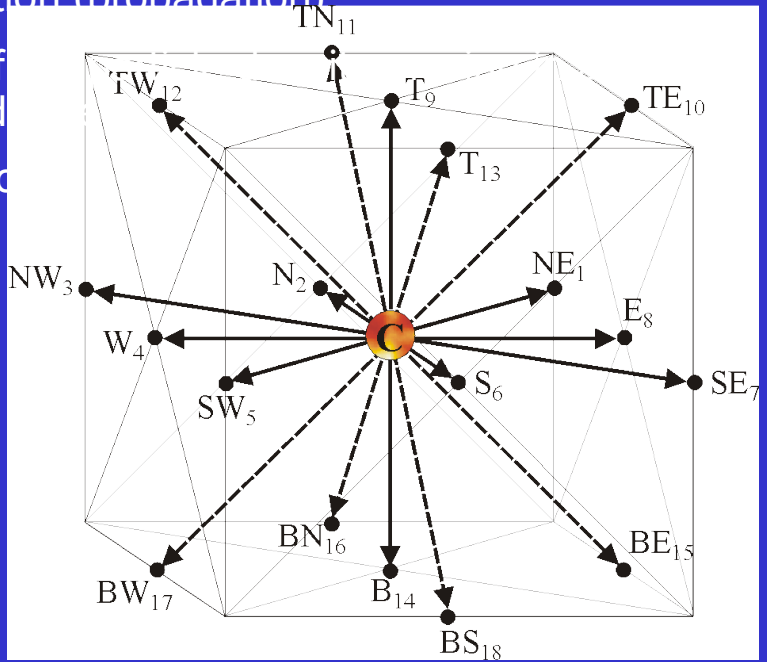


Lattice Boltzmann Method



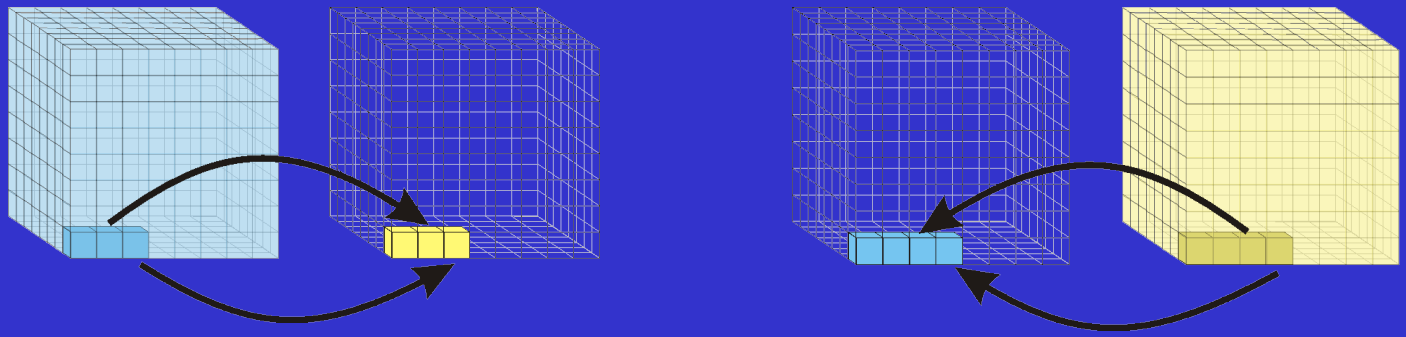


- The domain is discretised into „voxel“
- in each voxel *N* distribution functions f_i are stored representing particle's velocities
- from time step to time step, these distribution functions move to the next cell according to their microscopic velocity direction (propagation)
- the fraction of
  directions is d
- through the co
  approached

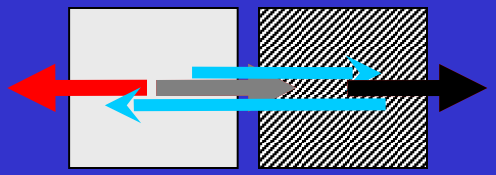


Lattice Boltzmann Method (Implementation Aspects)

- Two Grids/Compressed Grid due to data dependencies



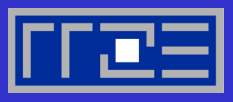
- Bounce Back at walls and obstacles



„bounce back“ at solids: particle distributions which would enter a solid cell are put back to the original cell with opposite microscopic velocity

Implementation Aspects

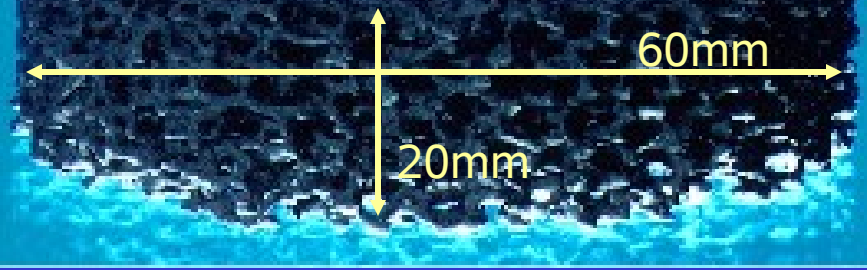
```
double precision f(0:xMax+1,0:yMax+1,0:zMax+1,0:18,0:1)
do z=1,zMax
   do y=1,yMax
      do x=1,xMax
         if( fluidcell(x,y,z) ) then
Collide:
            LOAD f(x,y,z, 0:18,t)
            Relaxation (complex computations)
Stream:
            SAVE f(x  ,y  ,z  , 0,t+1)
            SAVE f(x+1,y+1,z  , 1,t+1)
            SAVE f(x  ,y+1,z  , 2,t+1)
            SAVE f(x-1,y+1,z  , 3,t+1)
            …
            SAVE f(x  ,y-1,z-1,18,t+1)
         endif
      enddo
   enddo
enddo
```

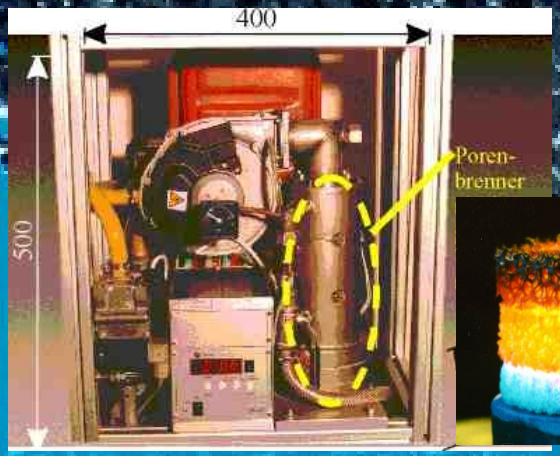


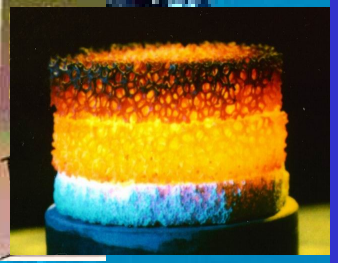
Application: Porous Media Combustion

Porous Media Combustion

- New technology for heating installations:
  Porous Media Combustion
  - Fuel-air-mixture does no longer react in a free flame
  - Combustion process takes place inside the pores of a porous medium that is placed in the reaction area

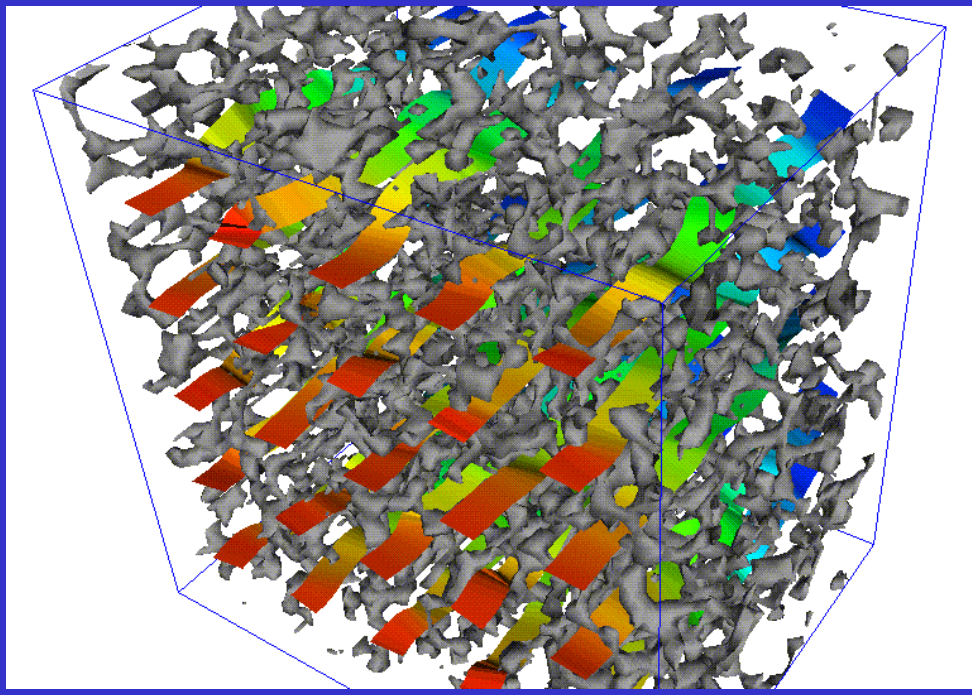






Application: Introducing Complex Geometries used

- Porous Medium from PMG: Silicon-Carbide (SiC)
  - Many small obstacles
  - Obstacle/fluid-ratio: ~2%
  - High number of fluid-solid faces

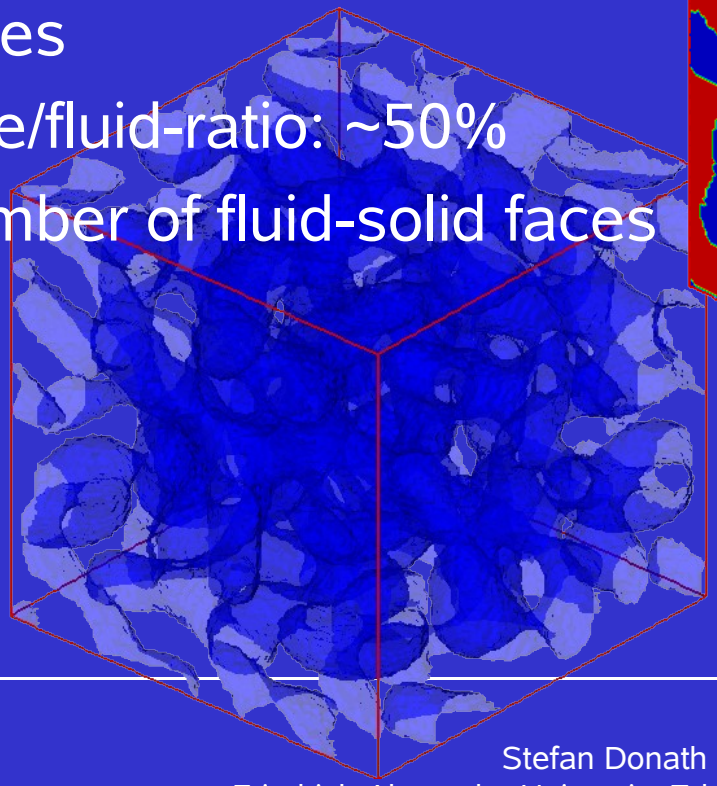


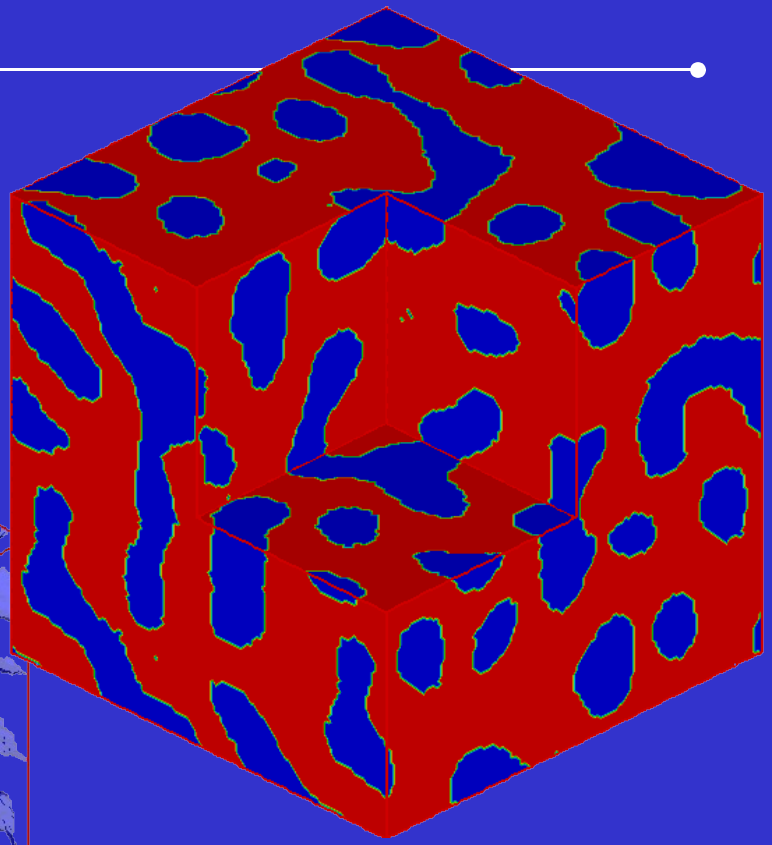
Application: Introducing Complex Geometries used

- Second Test-Geometry:

  "MC"

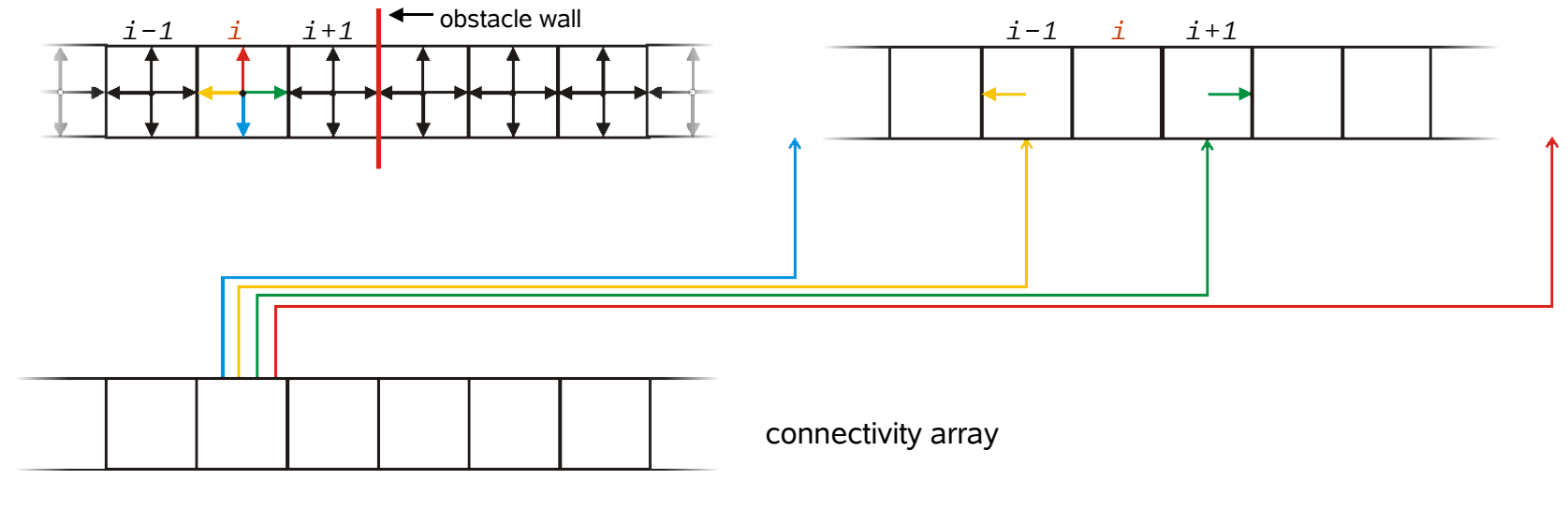
  - Huge obstacles, only few fluid tubes
  - Obstacle/fluid-ratio: ~50%
  - Low number of fluid-solid faces





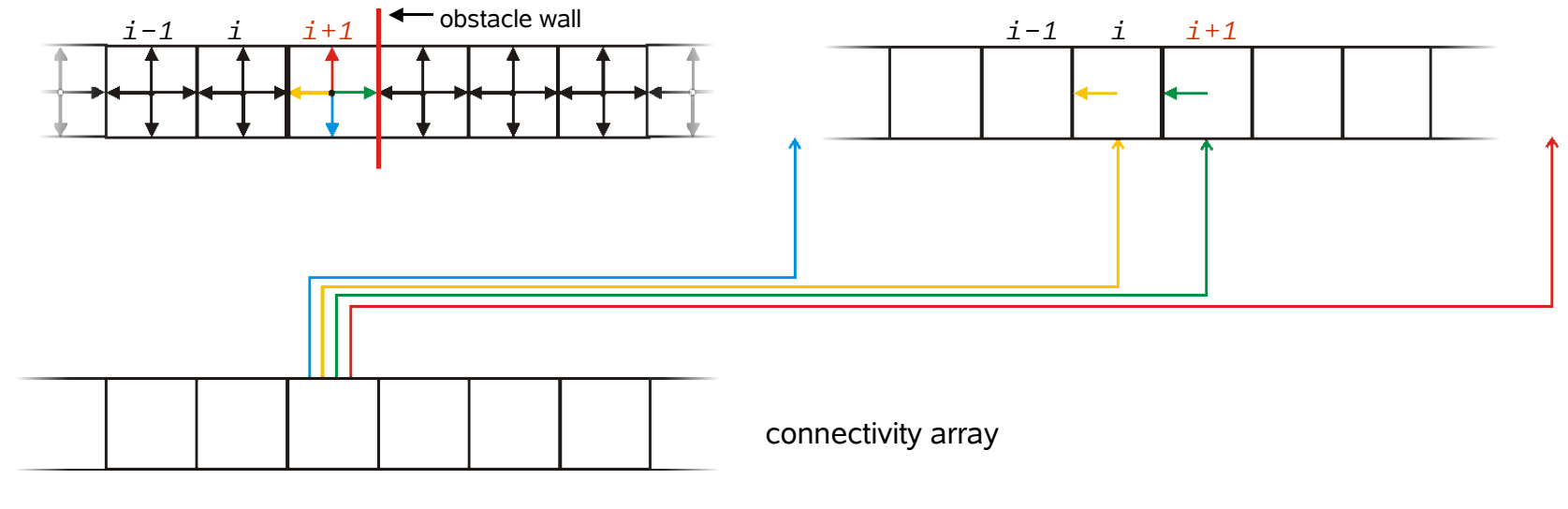
Implementation

- Indirect addressing and implicit Bounce Back



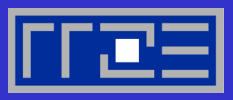
Implementation

- Indirect addressing and implicit Bounce Back



Implementation

- Preprocessor
  - Sets up connectivity array
  - Specifies all domain parameters:
    - Geometry and obstacles
    - Traversing scheme
- Solver
  - Reads in preprocessed information
  - Performs lattice Boltzmann method (in single loop)



Implementation

- Rating (1D-Array compared to standard implementation using multidimensional array and three loops):
  - Advantages
    - Saves memory

      The more obstacles in the domain the higher is the compression
    - Implicit Bounce Back

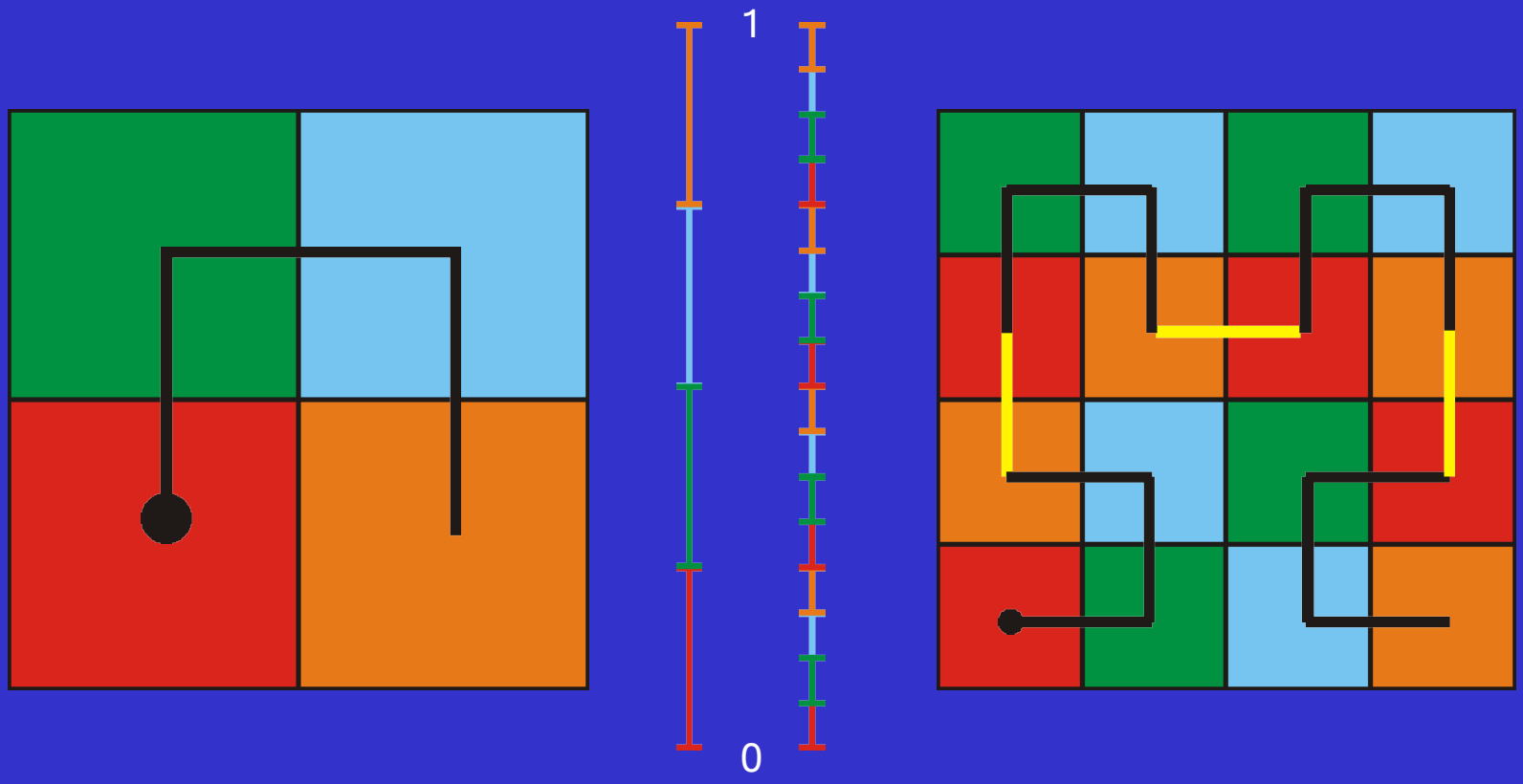
      No extra routine or if-statement needed
  - Drawbacks
    - Indirect addressing

      Prevents compiler from factorization and other optimization techniques

      ➔ Consequently, worse performance

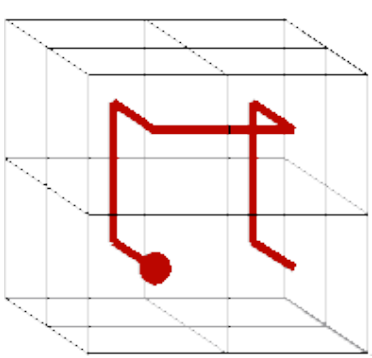
Memory Traversing Schemes: Space-Filling Curves

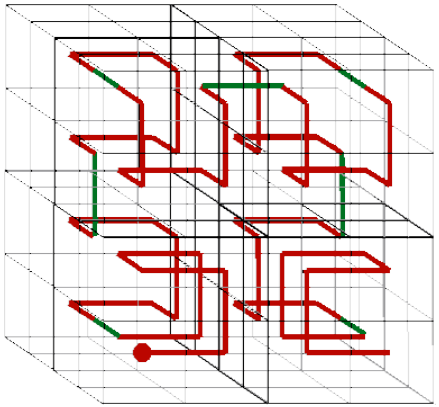
- Construction of Space-Filling Curves (Example: Hilbert in 2D)



Memory Traversing Schemes: Space-Filling Curves

- Table based approach
  - Hilbert: 48 Productions with 8 entries and 7 connectors
  - Peano: 8 Productions with 27 entries and 26 connectors

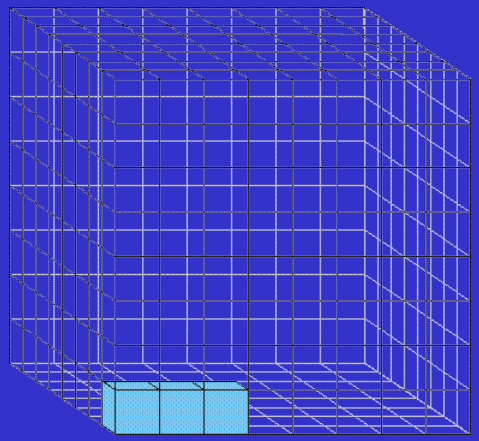




Current Level	Next Level														
bne	enb	b	ben	n	ben	f	fse	e	fse	b	bws	s	bws	f	wnf
fws	swf	f	fsw	w	fsw	b	bes	s	bes	f	fne	e	fne	b	nwb

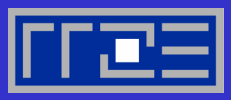
Memory Traversing Schemes: Blocking

- Implicit blocking technique
  - Arrangement of data in a blocking manner
  - Increases spatial locality



Memory Traversing Schemes

- Memory Traversing Schemes
  - Pure preprocessing technique:
    - Only arrangement of data in memory changed
    - No change for solver
    - No overhead for solver (e.g. loop overhead for blocking)
  - Space-Filling Curves:
    - Limitation in system sizes:
      - Hilbert: 2^{3n}
      - Peano: 3^{3n}
    - Enable mesh-refinement



Memory Layouts: Recalling Summary

- Standard Array-Layout: F(i,x,y,z,t)
  "Array-of-Structures", "Collision Optimized"

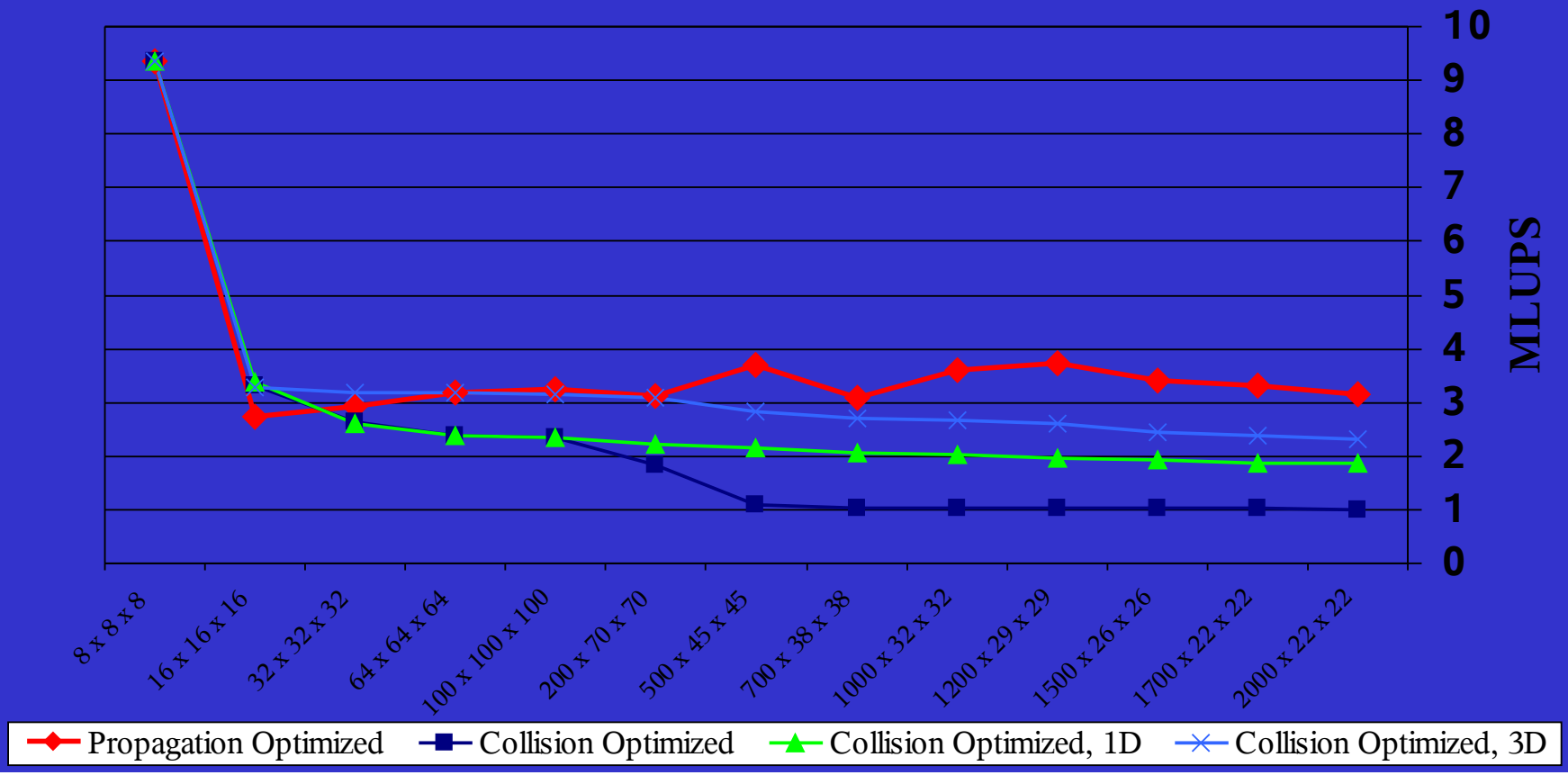
  - Collision optimized due to optimal read access (2 cache lines per Lattice Site Update (LUP) must be loaded)
  - Bad write access: Depending on system size the cache lines for the stores are in cache or not

- Optimized Array-Layout: F(x,y,z,i,t)
  "Structure-of-Arrays", "Propagation Optimized"

  - stride-1-access on x (inner loop)
  - 19 cache lines per 16 LUPs
    in read and write process
  - 1 cache miss each 16th memory

Memory Layouts: Performance Comparison

- Performance with simple solver (P4, 512kB L2)



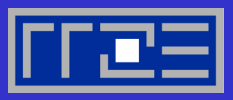
Further Optimization Techniques

- Additional Bottlenecks
  - Large loop body (causes register spills on IA32)
  - Concurrent writing to 19 different cache lines interferes with number of write combine buffers on IA32 (6 for Intel Xeon/Nocona)
  - Indirect addressing prevents IA32 hardware prefetcher from preloading values for target cells

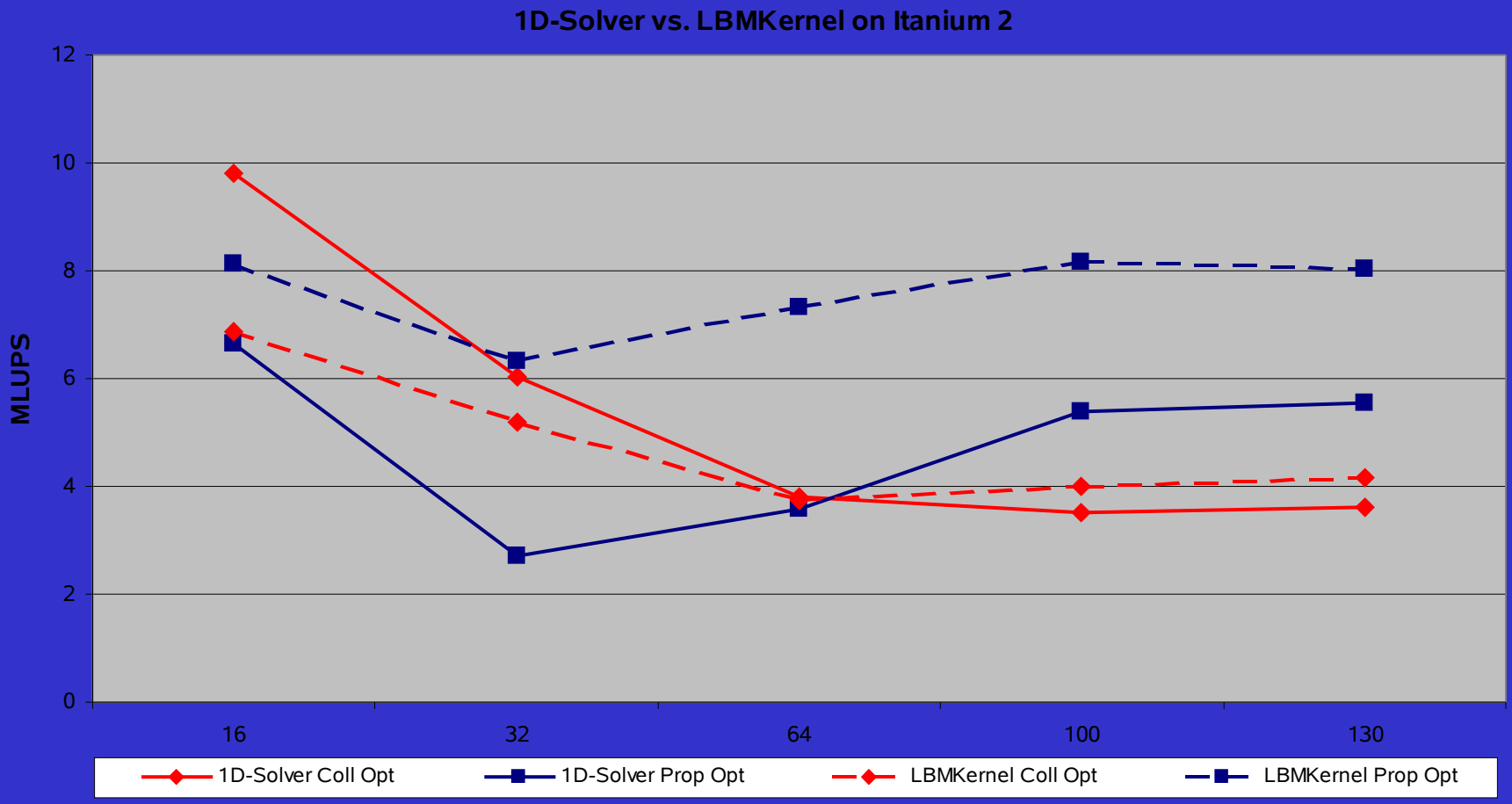
- Solutions (implemented in Solver):
  - Split up loop in 5 loops of length N_x
  - Manual Block Preload Technique

  (Drawback: Both techniques need a loop blocking scheme)

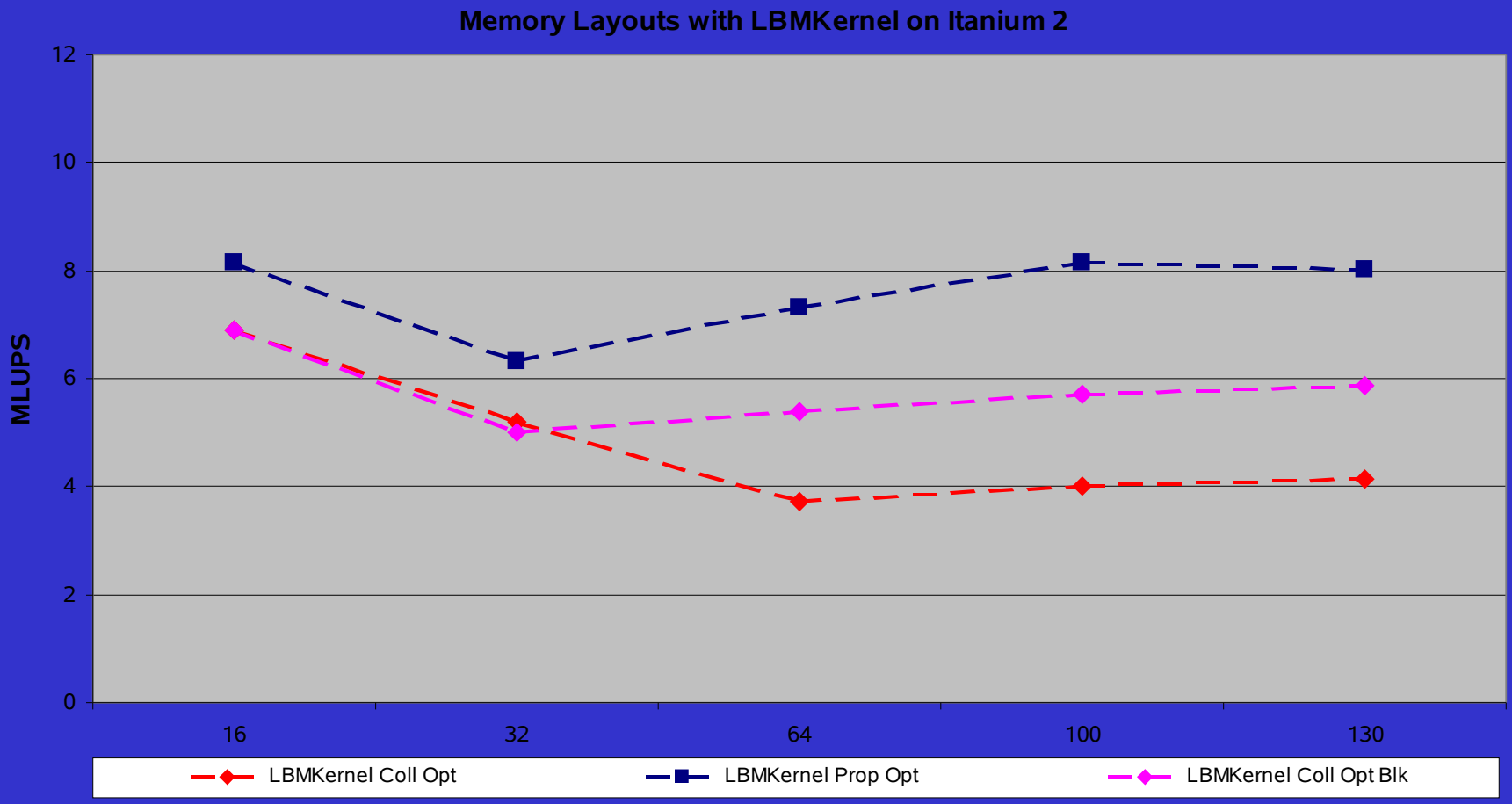
  These solutions only needed for IA32



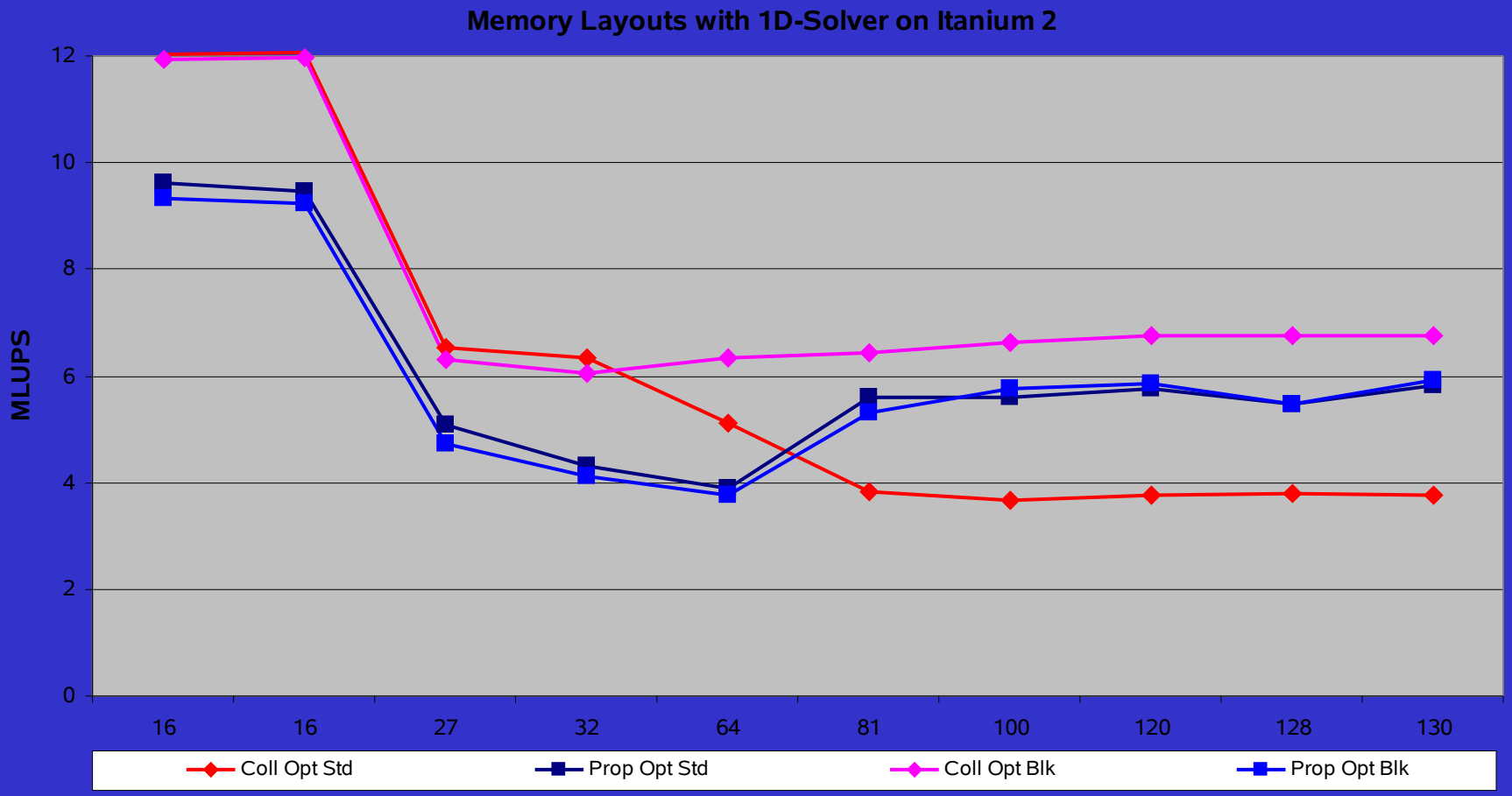
Comparison 1D-Array Solver to Standard Solver



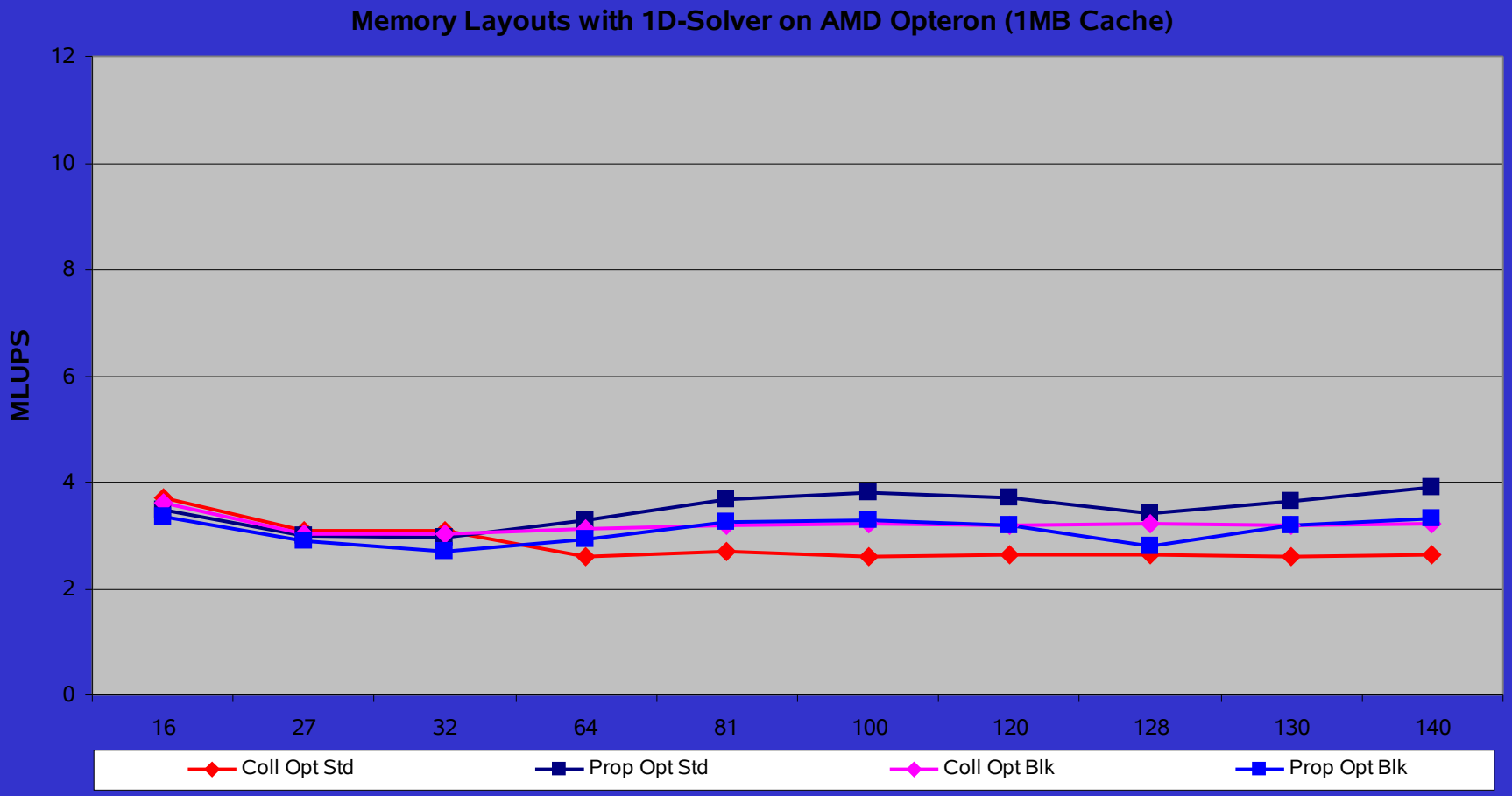
Memory Layouts for Standard Solver



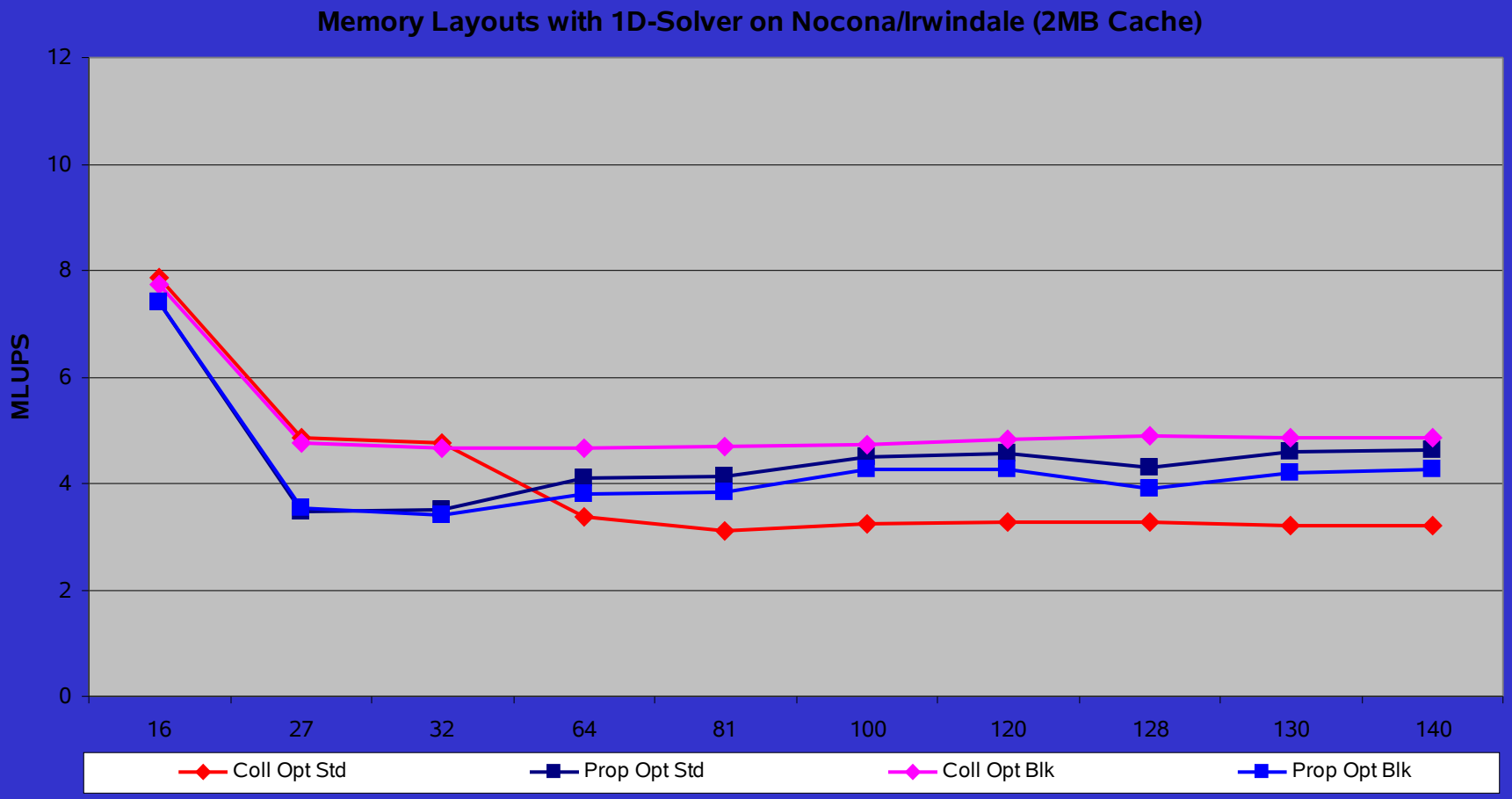
Memory Layouts on Itanium 2



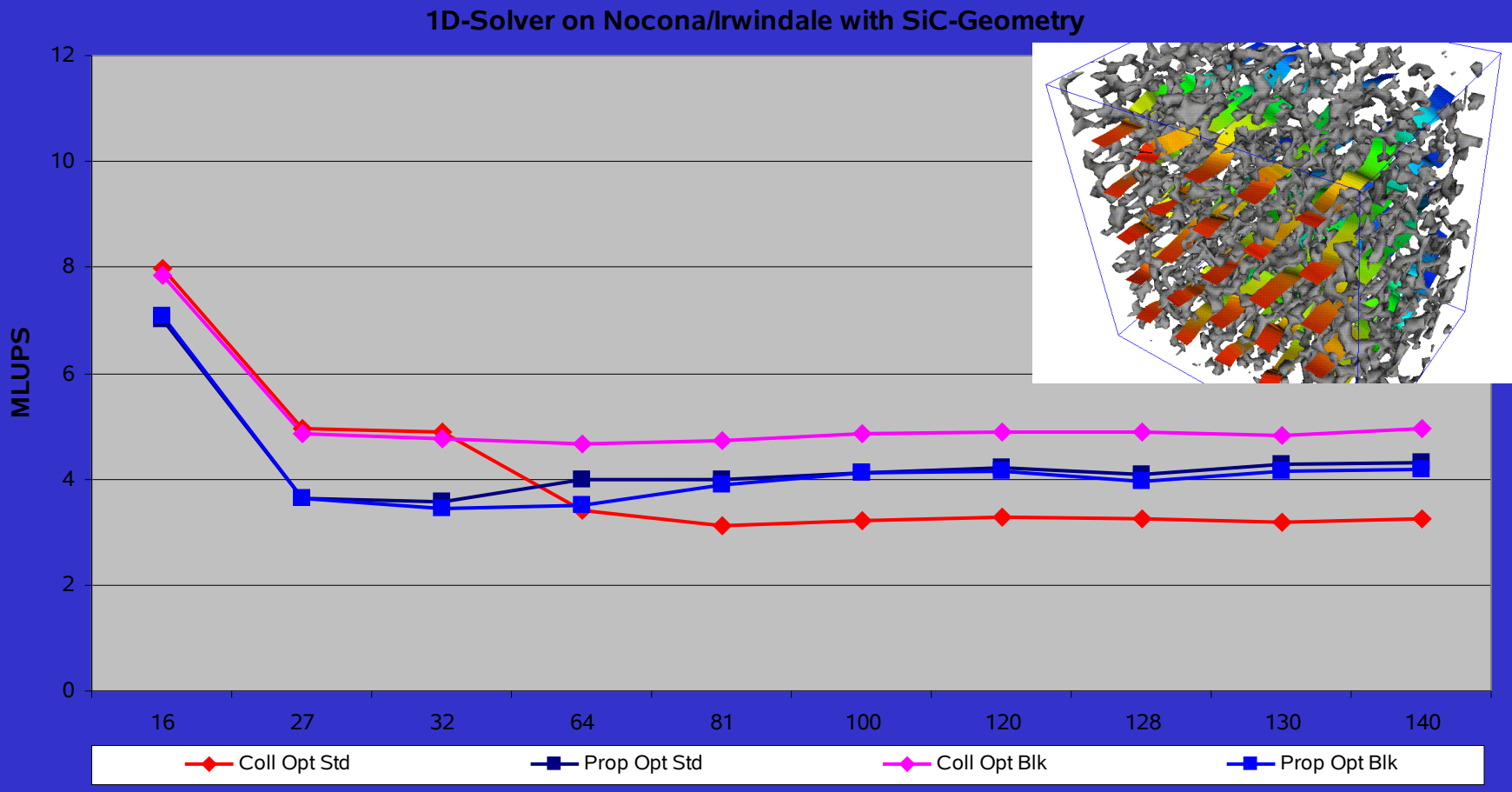
Memory Layouts on AMD Opteron (1MB Cache)



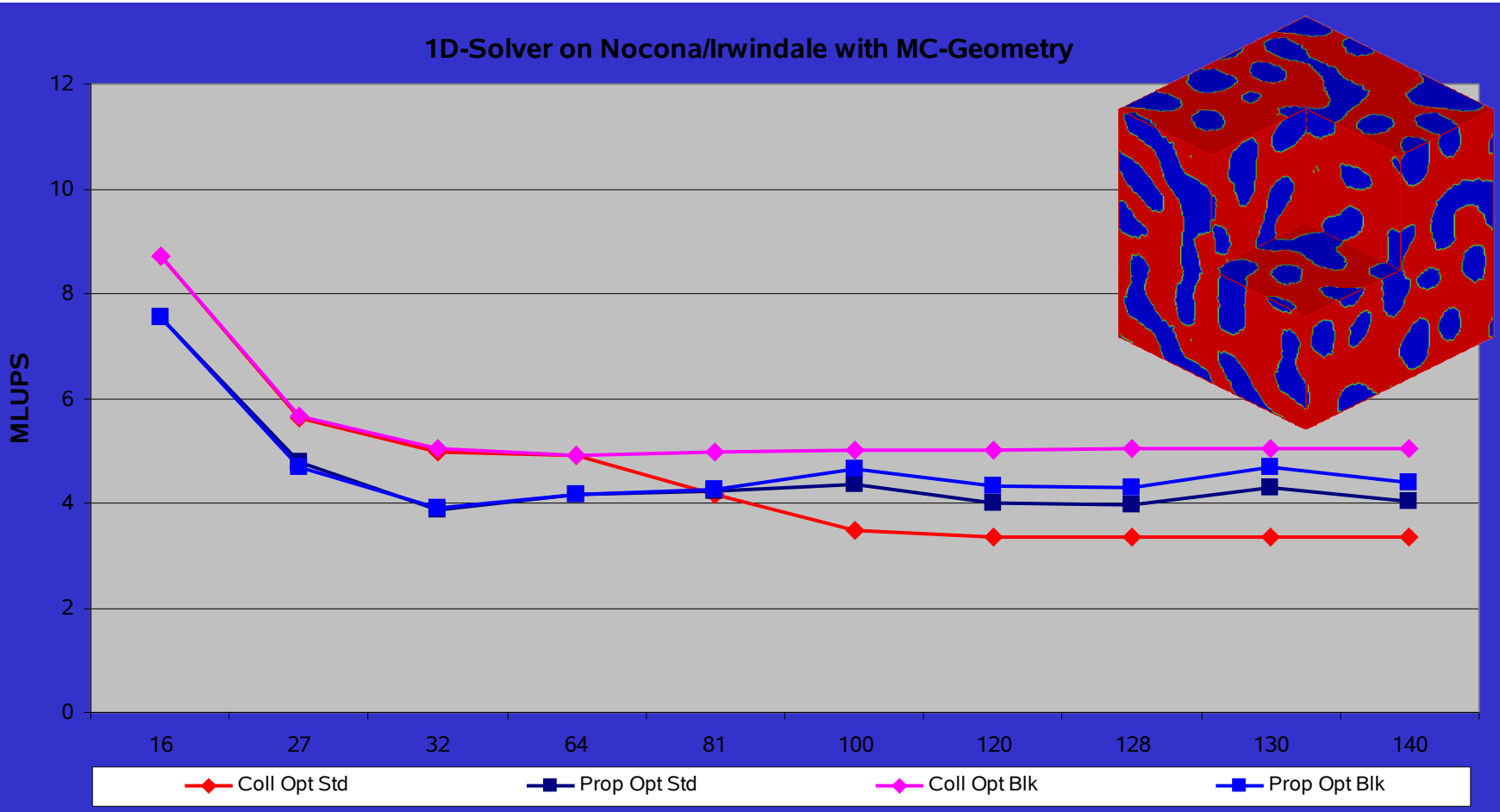
Memory Layouts on Nocona/Irwindale (2MB Cache)



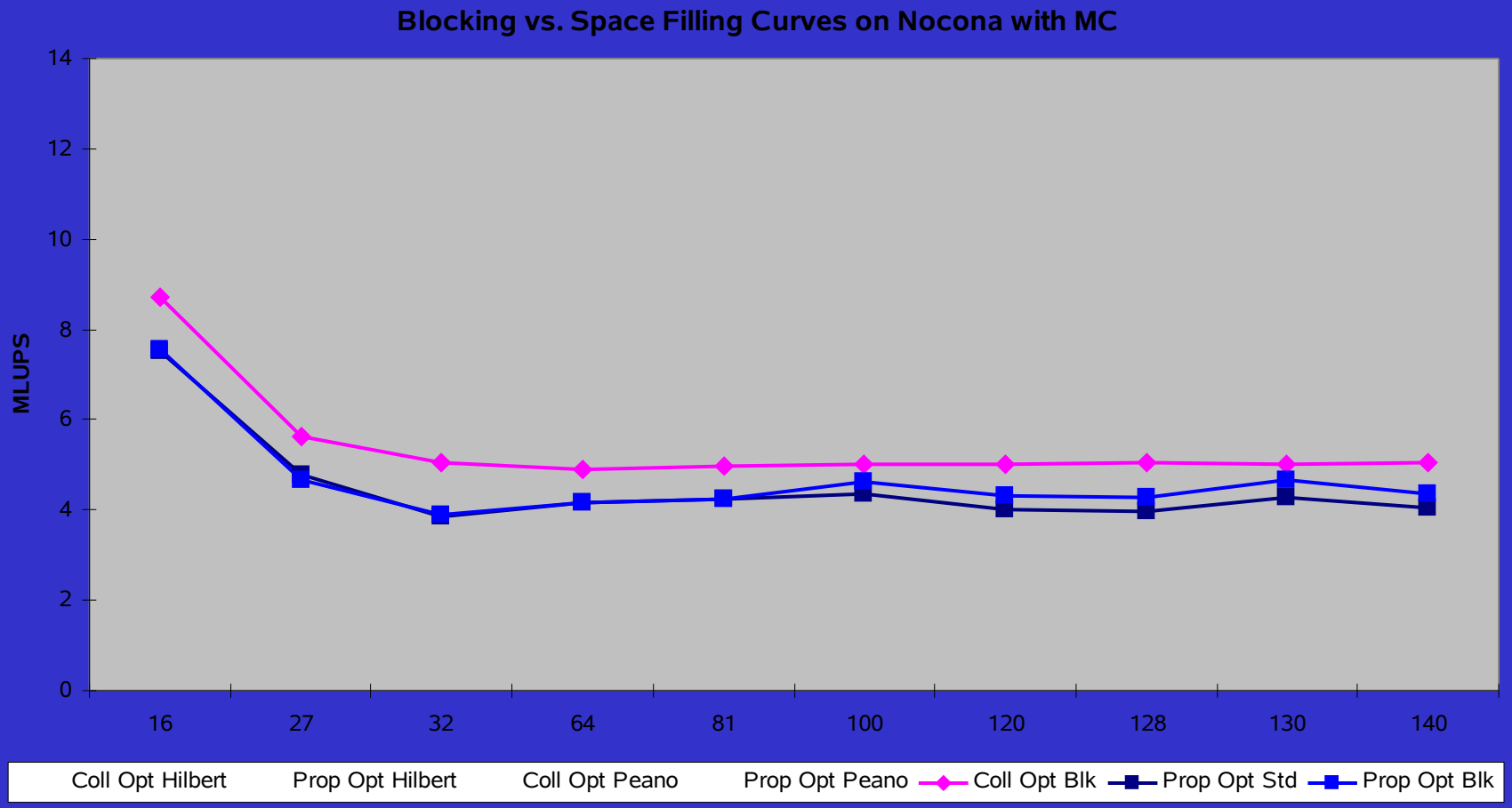
Influence of Geometry: SiC-foam (~2% obstacles)



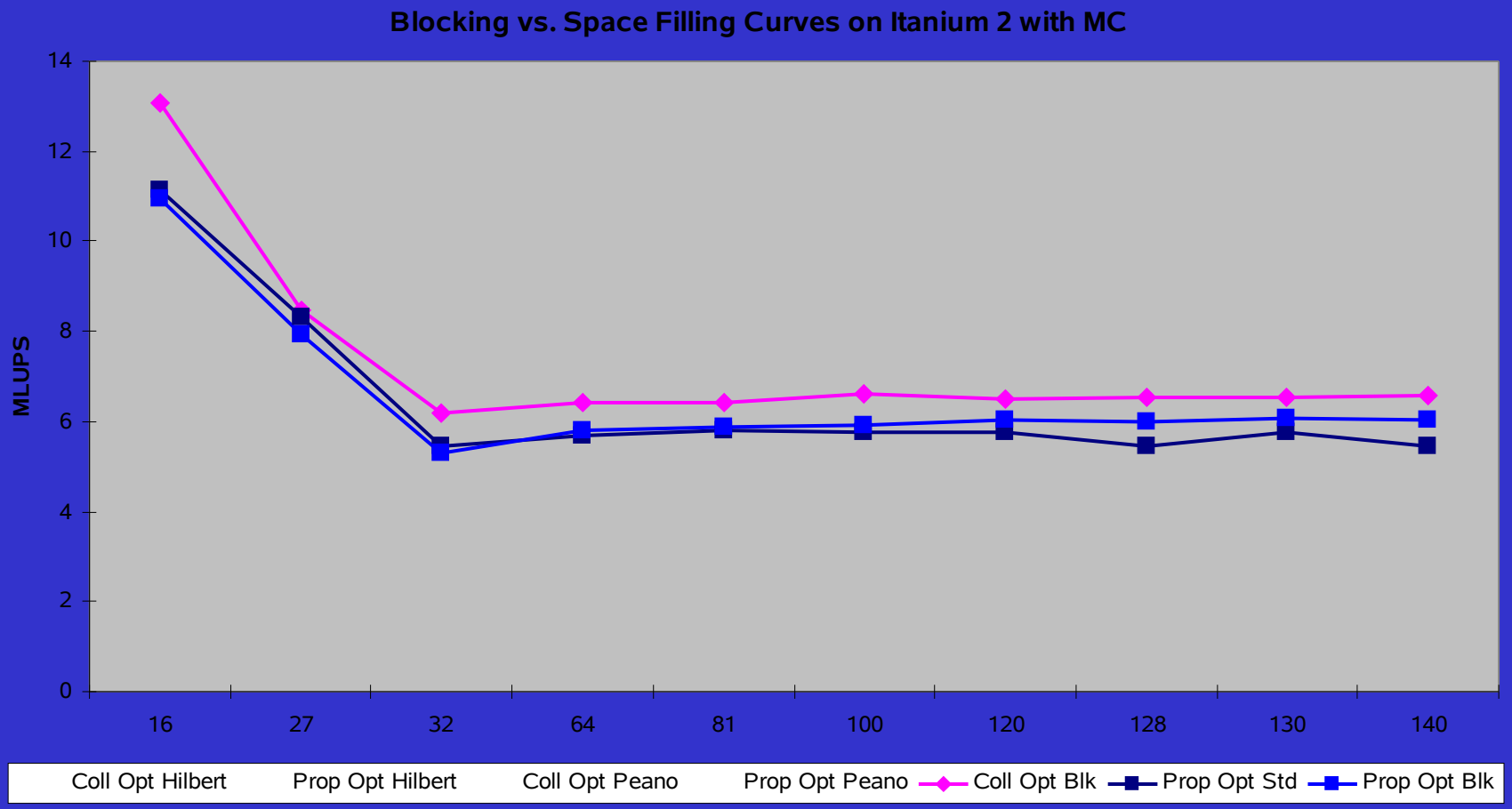
Influence of Geometry: MC (~50% obstacles)



Memory Traversing Schemes

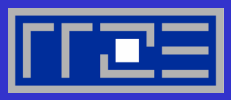


Memory Traversing Schemes



Conclusion

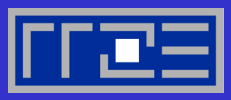
- 1D-Array Data representation makes performance independent of obstacle to fluid ratio
- Memory traversing by Space-Filling Curves results in similar performance as spatial blocking
  - ➔ Implementation of SFCs is not worth the effort (if they are used as memory traversing alternative only)
- Together with indirect addressing Collision Optimized Layout with blocking is best technique if cache is larger than 1 MB
- Indeed, there are cases where Propagation Optimized Layout is not best



Outlook

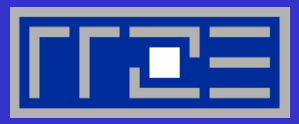
- Ongoing research:
  - Clarify why Itanium 2 suffers from indirect addressing
  - Resolve Propagation Optimized Layout‘s problems

- Future work could concern:
  - Space-Filling Curves:
    - Kind of staggered SFCs, for every direction own curve
      - ➔ Avoid waste of underused cache lines where lattice sites are neighboring cells which are visited much later
    - Galerkin-discretization or point wise evaluation of LBM to enable stack-implementation in conjunction with SFCs
    - BUT: For real-world problems construction on non-cubic grids is necessary at first



References / Thanks

THANK YOU!



HPC@Regional Computing Center of Erlangen (RRZE)
http://www.rrze.uni-erlangen.de/dienste/arbeiten-rechnen/hpc/

Acknowledgement:

- Prof. Dr. Ulrich Rüde (LSS)
- Bavarian Graduate School of Computational Engineering / ENB
- Co-Authors:
  Thomas Zeiser, Dr. Gerhard Wellein, Georg Hager, Johannes Habich

Stefan Donath:
stefan.donath@rrze.uni-erlangen.de

Memory Layouts: Collision Optimized Layout

- 18 write accesses on one cell from 3 different z-layers

$$Mem_k \approx 3 \cdot (N_i + 2) \cdot (N_j + 2) \cdot ¿$$
$$¿ \quad \cdot 2 \cdot 19 \cdot 8 \ Byte$$

- 8 write accesses on one cell from 3 different rows

$$Mem_j \approx 3 \cdot (N_i + 2) \cdot ¿$$
$$¿ \quad \cdot 2 \cdot 19 \cdot 8 \ Byte$$