# Optimizing Numerical Simulation Kernels for Current Instruction and Memory Architectures

*ScalPerf 05*

Jan Treibig, Ulrich Ruede (Erlangen)

Ben Bergen (Los Alamos)

*October 2005*

www10.informatik.uni-erlangen.de

# Overview

1. Motivation
   - Trends in modern architectures
2. Optimization techniques
   - Influences on performance
   - Systematic approach to optimization
3. Examples
   a) Optimization of the Red-Black Gauss-Seidel Algorithm
   b) Optimization of the Lattice Boltzman Algorithm
   c) Excursion: Unstructured meshes
      - Problems with unstructured meshes
      - HHG Results
4. Summary

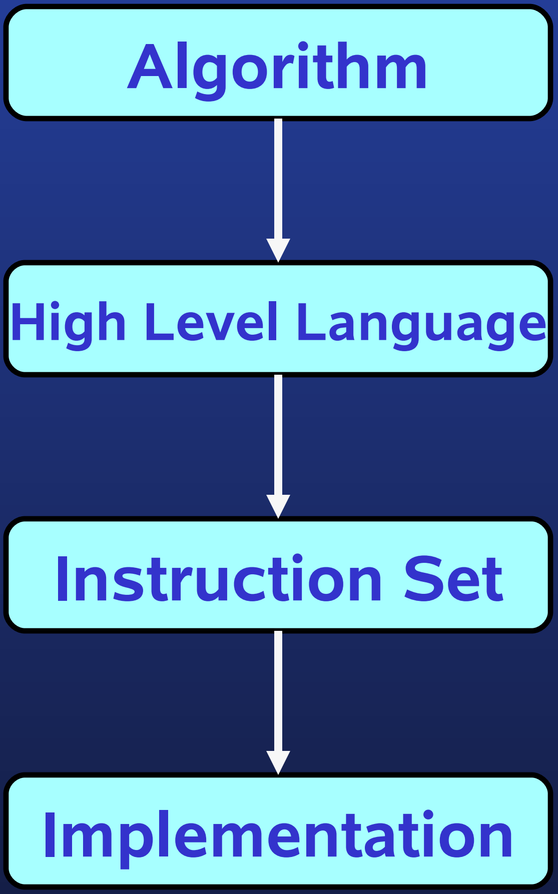
# Trends in modern architectures

- Hard to utilize full potential of a CPU
- Observation on modern architectures:
  - Hardware features are not fully exploited by current compilers
  - Instruction set issues limit performance severly
- There is a trend to explicit parallelism either in the instruction set (e.g. IA64) or the high level language (e.g. multi core), Cell processor

Problem:

New developements in hardware design seem to be difficult to exploit for compilers. Therefore the gap between the ISA and the capabilities of the compiler seems to widen

# Layers of the optimization process



Algorithm

High Level Language

Instruction Set

Implementation

- Performance is generated in the implementation only
- Each layer can destroy the result
- All layers should be designed in an integrated fashion

# Systematic Approach to Optimization

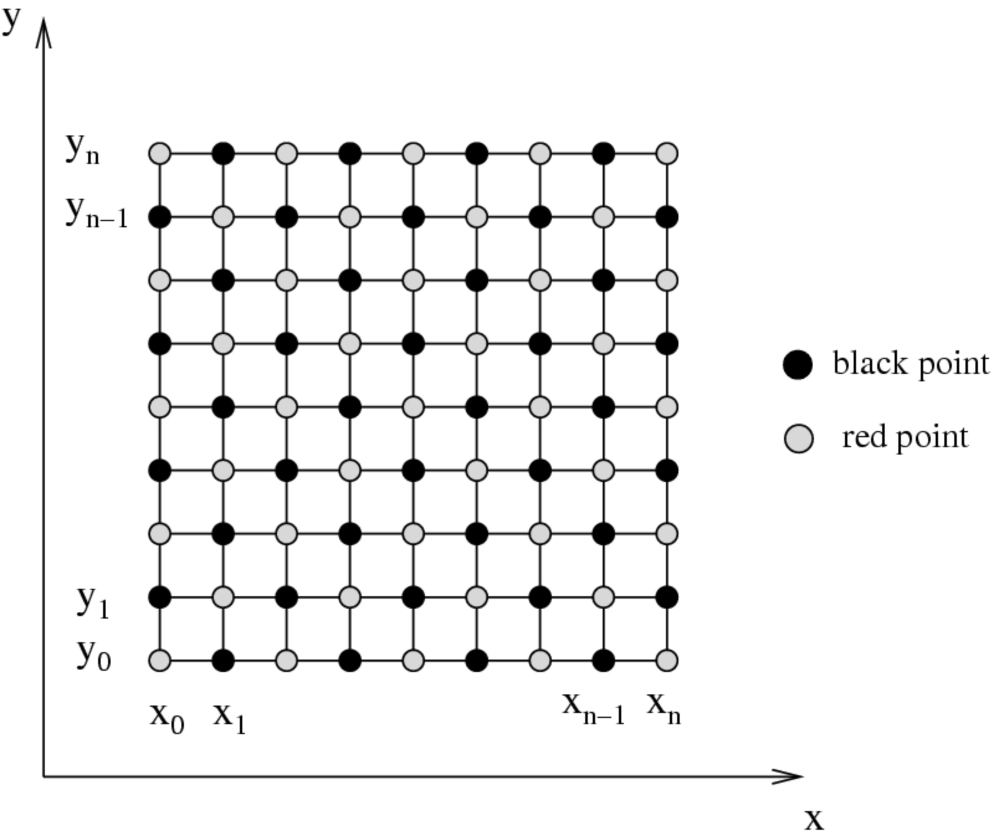
We are interested in loop based, initially memory (bandwidth) limited, iterative algorithms

- Find a sensible upper bound of reachable performance
  - Develop benchmarks to explore capabilites of target architecture
  - Define metrics to compute theoretical upper performance bounds
- Reachable is what the implementation is capable of

Consequence:

In some cases it might be inevitable to implement the algorithm directly in terms of the instruction set to efficiently use the potential of the CPU.

# Part 3a)
# Red-Black Gauss-Seidel Algorithm



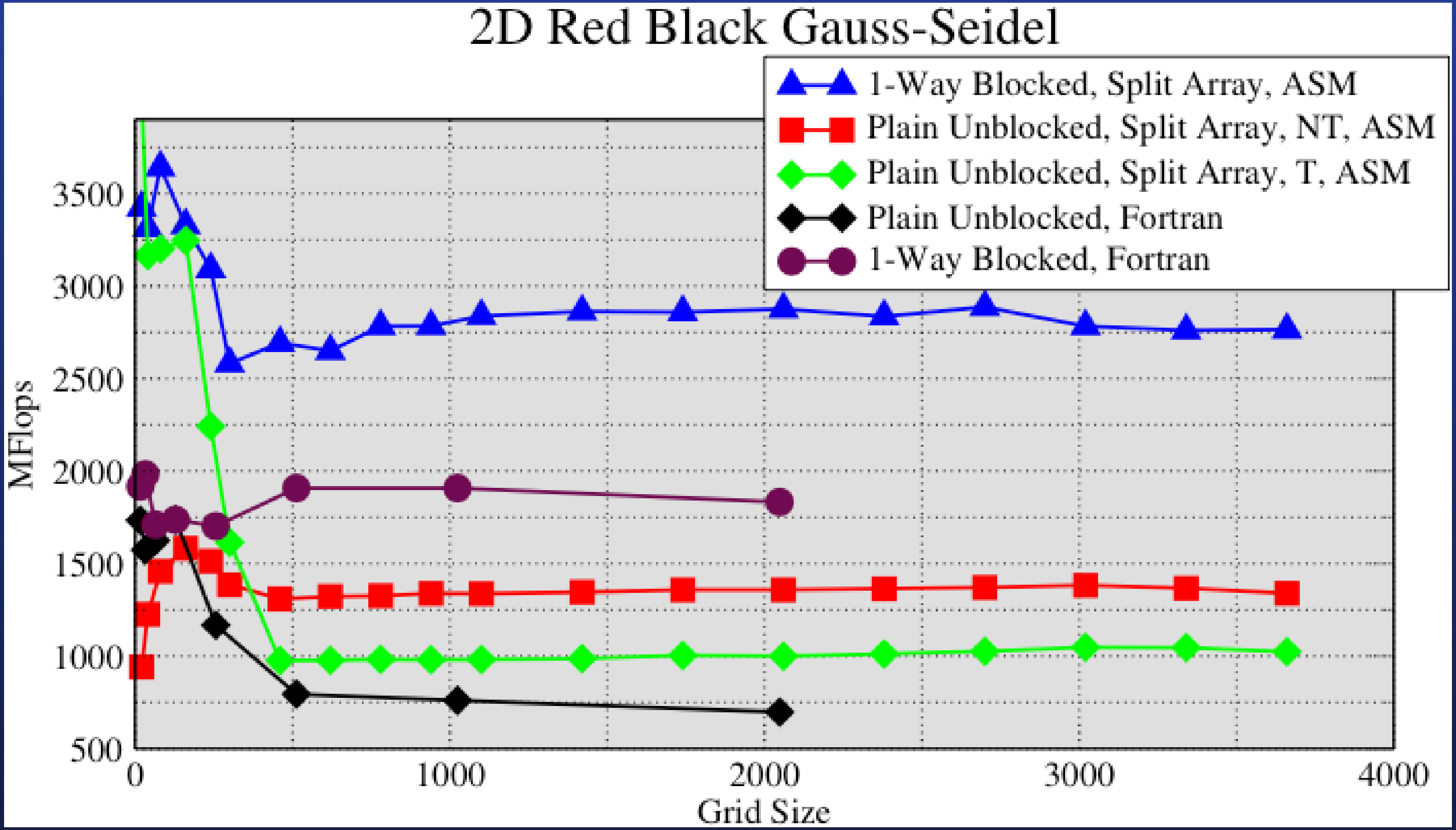
# Optimization of the Red-Black Gauss Seidel Algorithm

Target architecture: x86 (SSE3 extension)

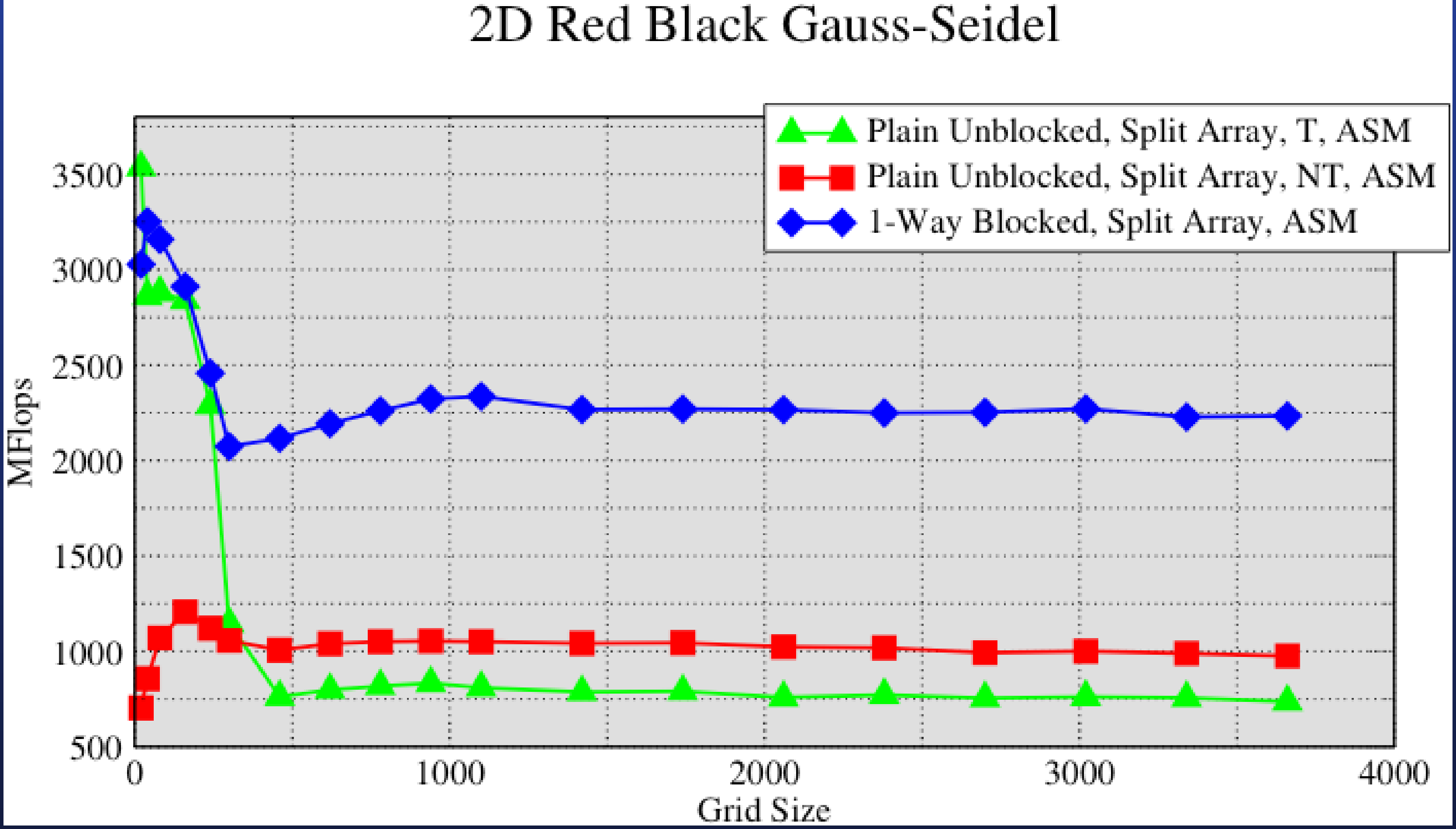
Target implementation: Intel Pentium 4 Prescott.

- Split up data layout to enable 16 byte moves and SIMD (SSE) instructions
- Introduce special cache blocking techniques (Dime project)
- Instruction scheduling and optimized addressing

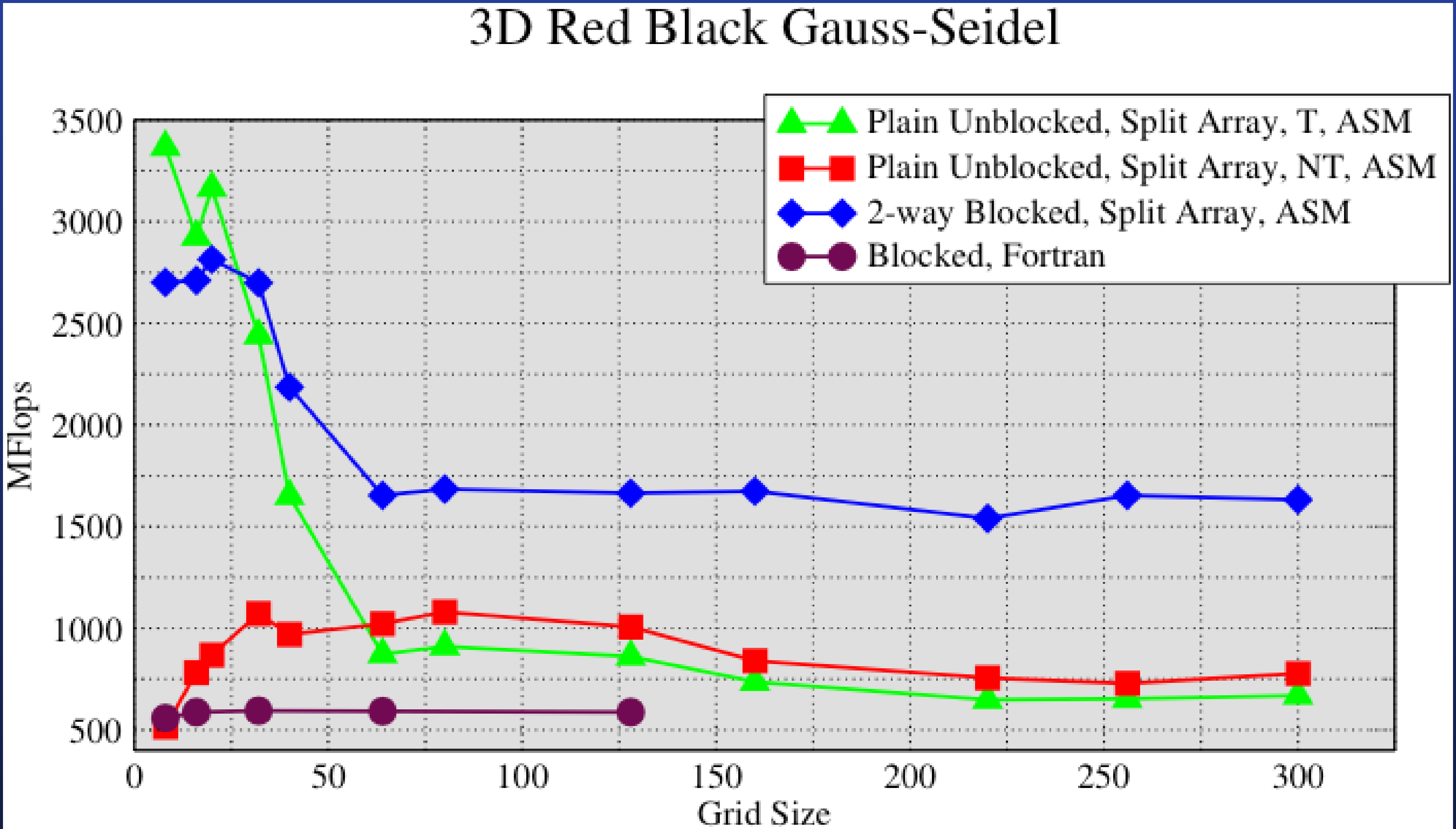
# RB-Gauss Seidel: Results 2D
## (with coefficients)



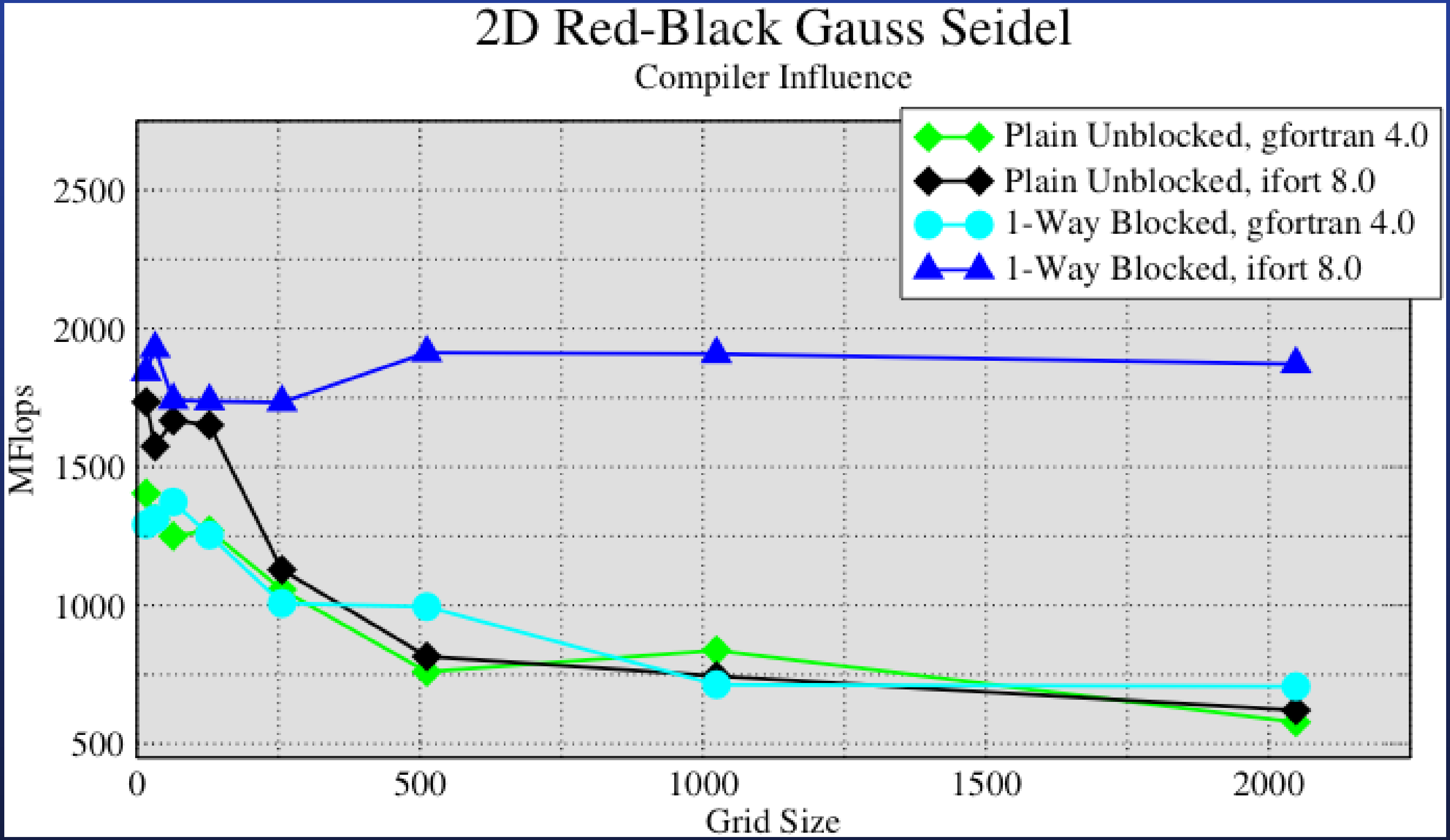
# RB-Gauss Seidel: Results 2D
## (w/o. coefficients)



# RB-Gauss Seidel: Results 3D



# RB-Gauss Seidel: Compiler Comparison



# RB-Gauss Seidel

## (Compiler influence cont‘d)

| | gfortran 4.0 | ifort 8.0 |
|---|---|---|
| Total instructions | 132 | 67 |
| Integer instructions | 70 | 6 |
| Stack accesses | 10 | 0 |
| Total Cycles | 303 x 10^6 | 120 x 10^6 |

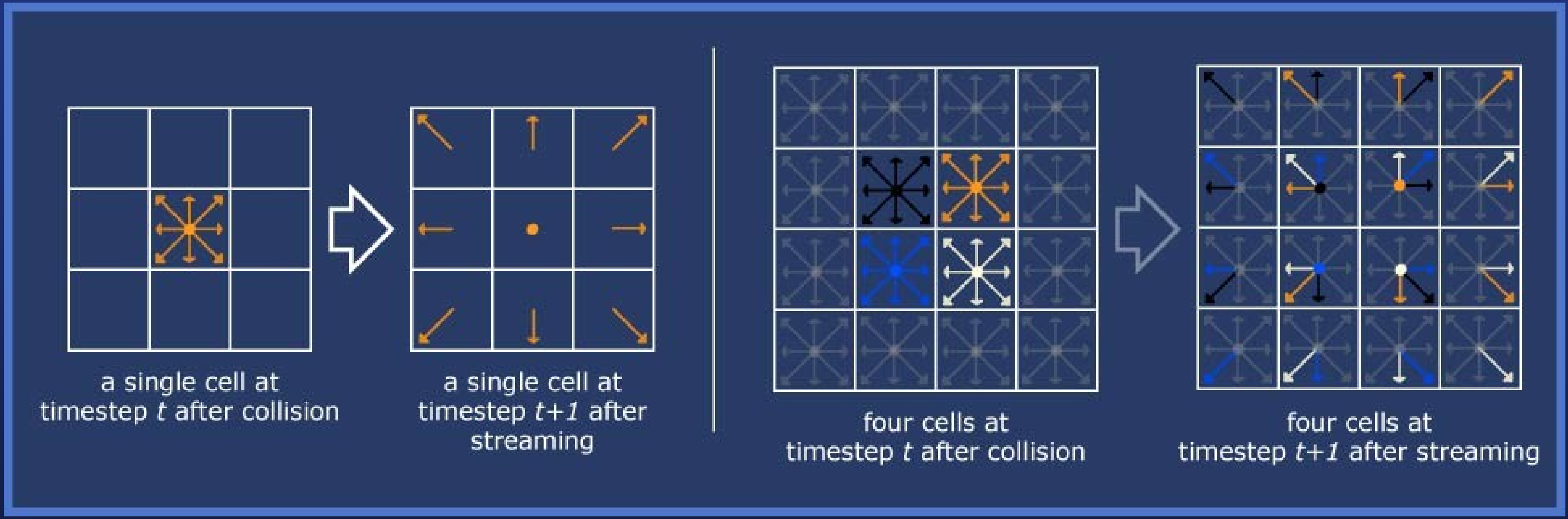
- The compiler may destroy the effect of the hand optimiziation by generating inefficient code
- Addressing is an elementary task of the compiler, but

  → Never trust an compiler!

# RB-Gauss Seidel: Analysis

- After applying common cache blocking techniques instruction set issues limit performance
  - efficient use of SSE vs. FPU
  - nontemporal moves
  - prefetching
- Especially on x86 (?), compilers are rarely capable of generating fully efficient code for complicated nested loop structures (as generated by cache-blocking)
- standard FORTRAN implementation vs. hand coded: Improvement by almost factor 5!

# Part 3b)
# Lattice-Boltzmann Algorithm for Computational Fluid Dynamics



a single cell at timestep *t* after collision

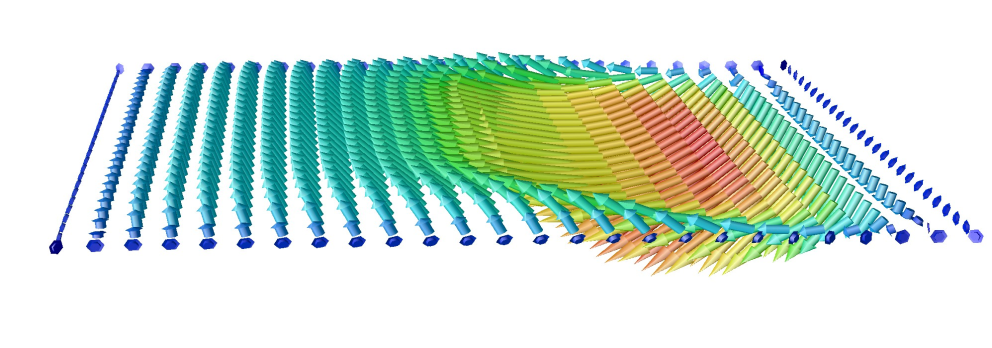
a single cell at timestep *t+1* after streaming

four cells at timestep *t* after collision

four cells at timestep *t+1* after streaming

# Optimization of the Lattice Boltzman Algorithm

- The Lattice Boltzmann method is an incompressible fluid flow solver based on a cellular automaton approach
- Time is spent almost exclusively in one loop body
- The inner loop body consists of about 160 flops
- There are 19 loads and 19 stores per update in several streams



Zur Anzeige wird der QuickTime Dekompressor YUV420 codec bentigt.

# Lattice Boltzman: Optimizations

Architecture: x86-64

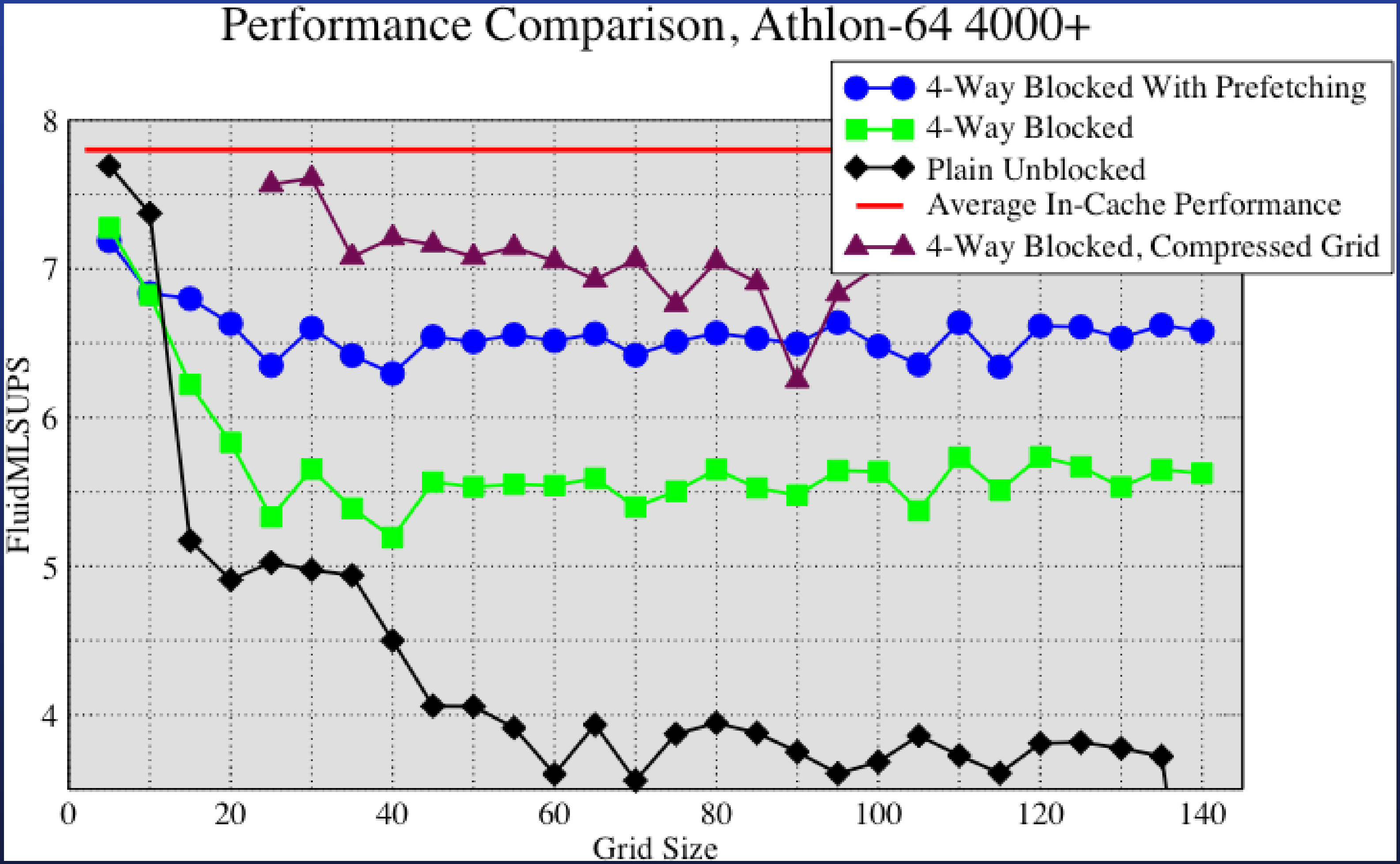
Target implementation: AMD Athlon 64

Code written in x86-64 assembler

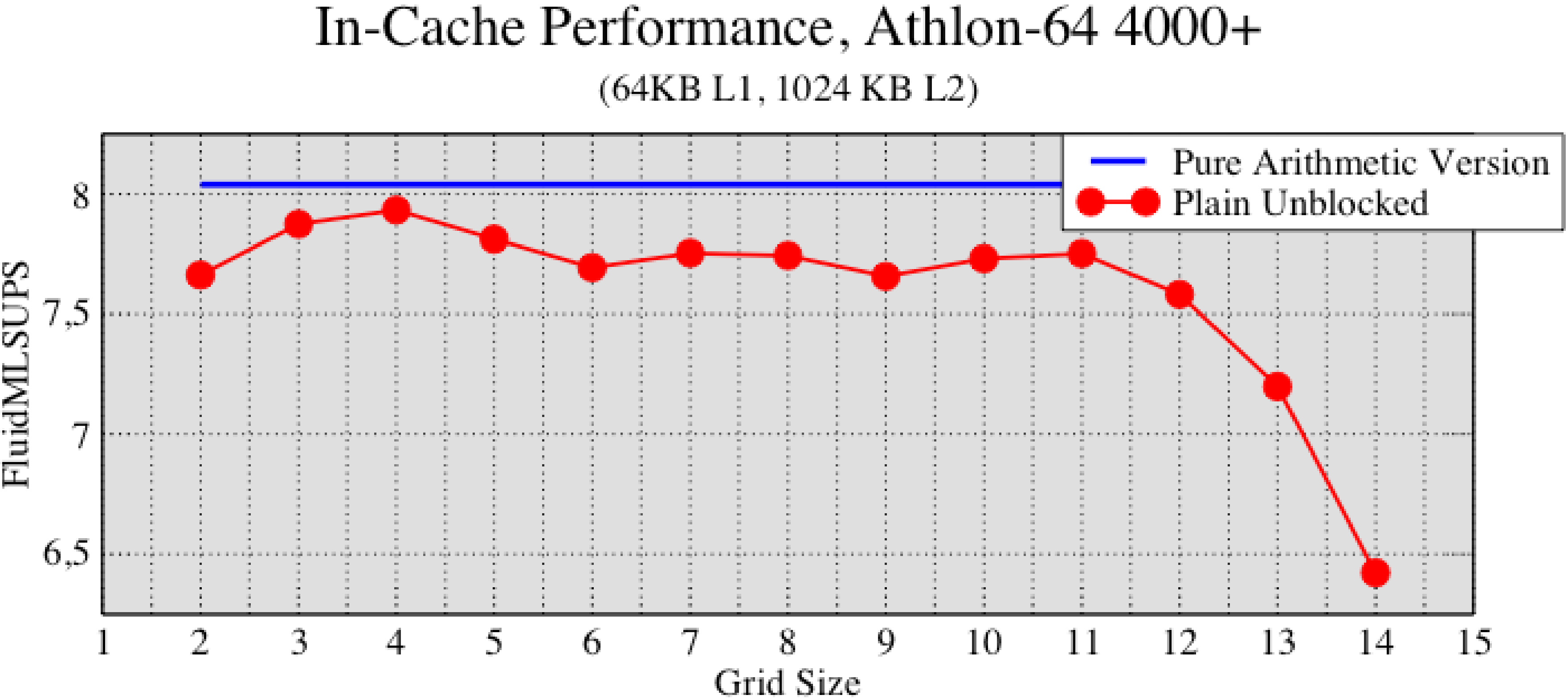
Techniques applied:

- SIMD Vectorization
- Software prefetching
- 4-way blocking techniques

# Lattice Boltzman Results: Memory



# Lattice Boltzman Results: In Cache



# Lattice Boltzman: What is Going On?

- After applying cache blocking techniques and release pressure from the memory bus, arithmetical limitations (dependencies) are dominating
- Instruction scheduling is difficult for a code of that size (for humans?)
- When combining vectorization with better instruction scheduling, better performance should be possible
- Theoretically, neither the memory bus nor the FPU are saturated - there should be room for further improvement.

# Part 3c)
# Excursion: Sparse Matrix Computations

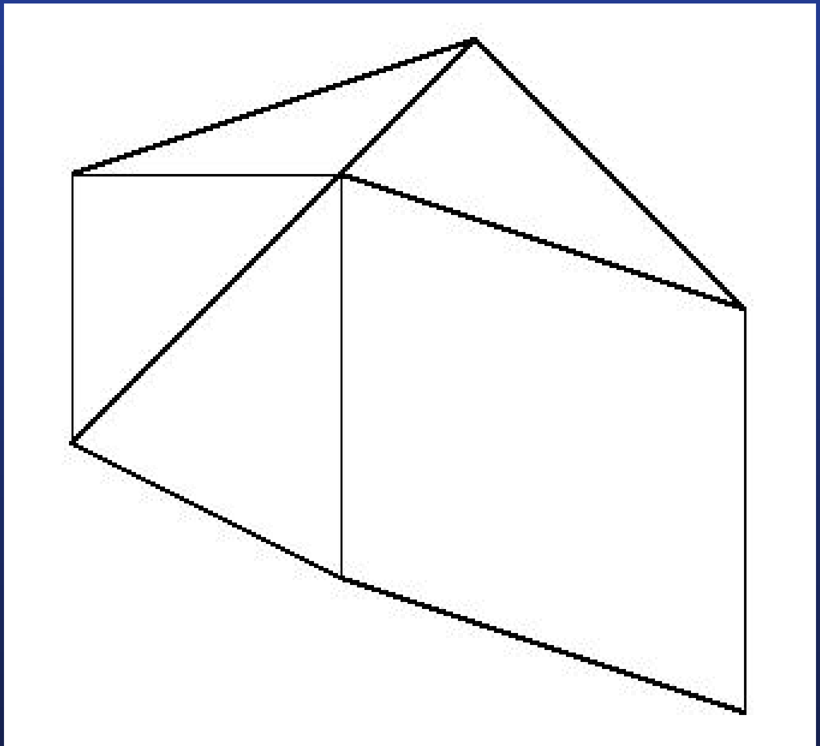
# Reasons for poor efficiency of standard sparse matrix data structures

- Cache Effects
  - Difficult to find an ordering of the unknowns that maximizes data locality
- Indirect Indexing
  - Precludes aggressive compiler optimizations that exploit instruction level parallelism (ILP)
- Variable Coefficients
  - Overkill for certain class of problems

# Hybrid Hierarchical Grids: Basic Concepts

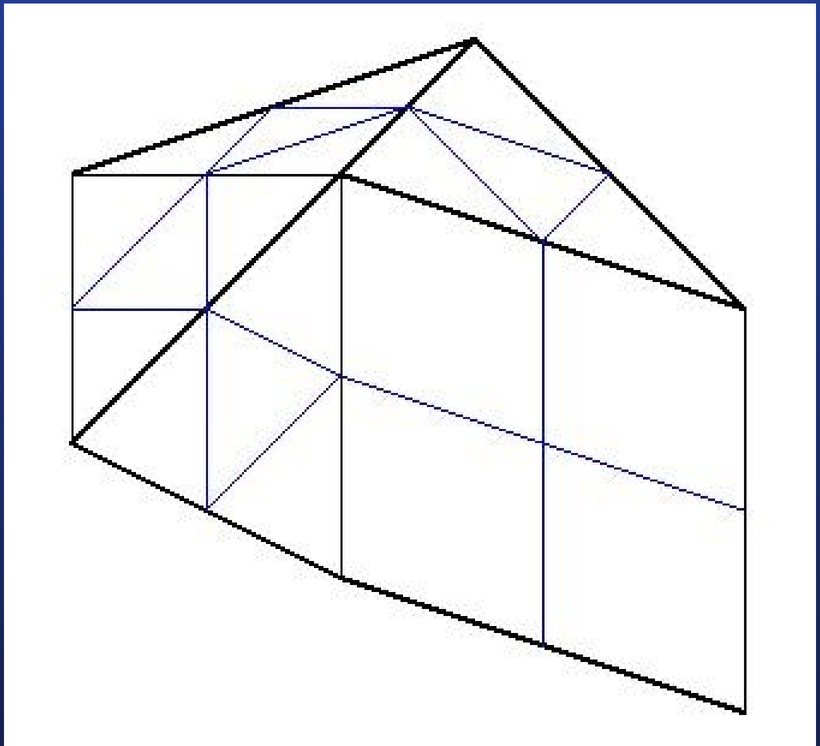
- Purely unstructured input grid
- Input grid resolves large scale, structural features of the problem domain patch-wise constant coefficient
- Apply patch-wise, regular refinement to each element of the input grid
- Patch interiors are structured and constant coefficient
- Generates nested grid hierarchy suitable for geometric multigrid

# Refinement example



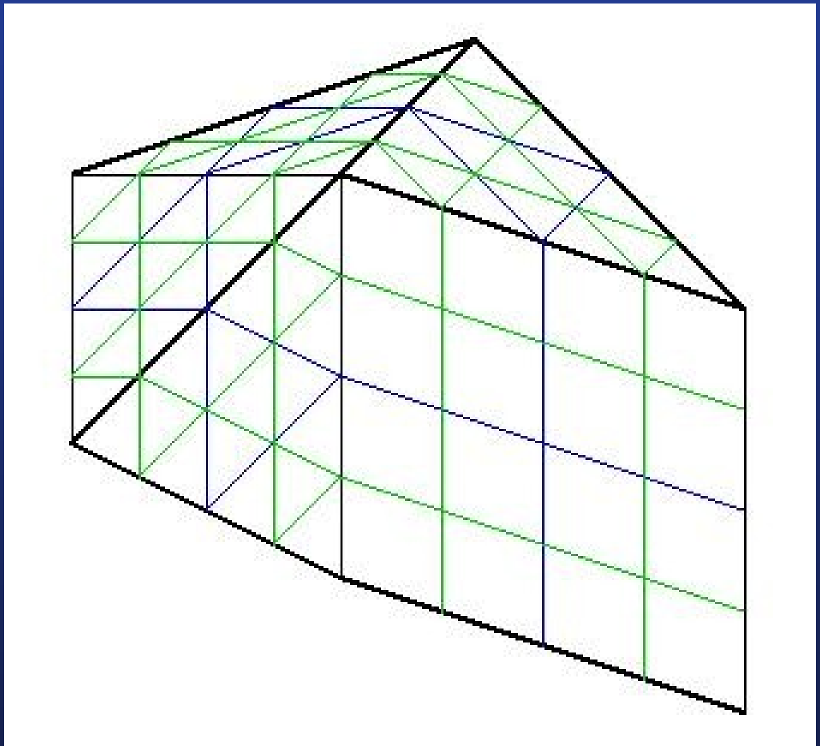
Input Grid

# Refinement example



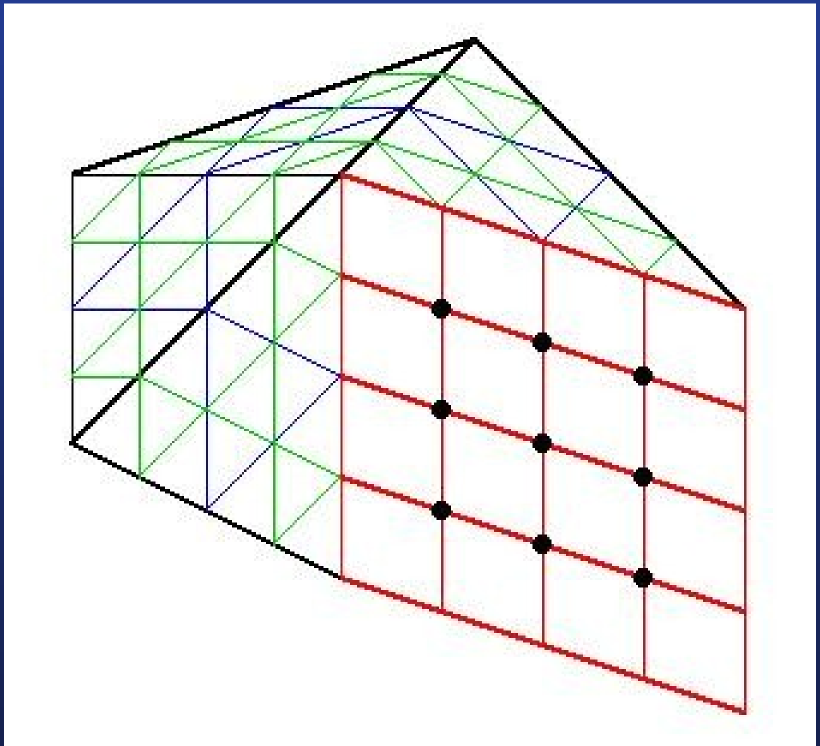
Refinement Level one

# Refinement example



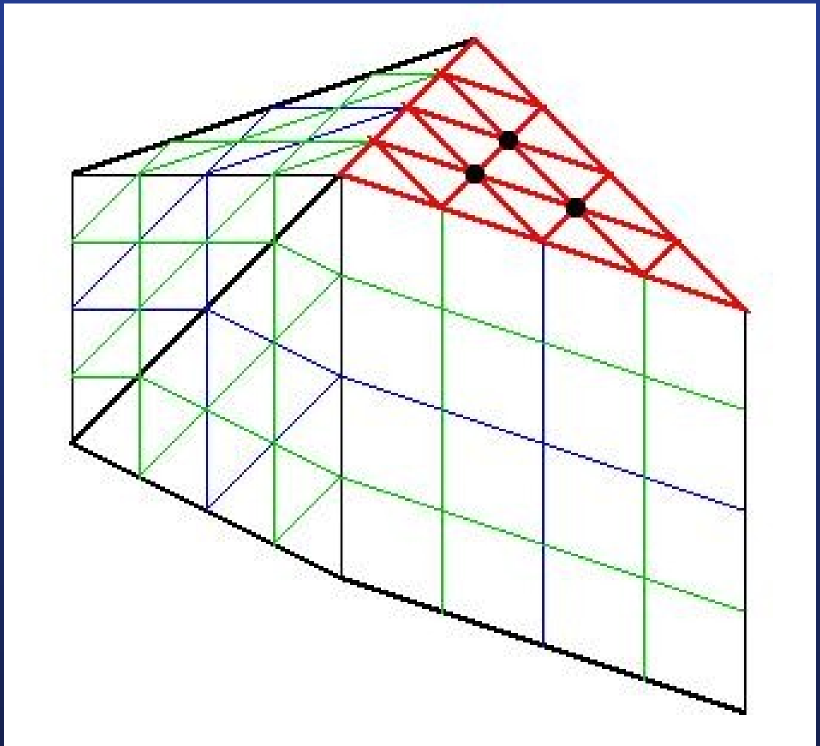
Refinement Level Two

# Refinement example



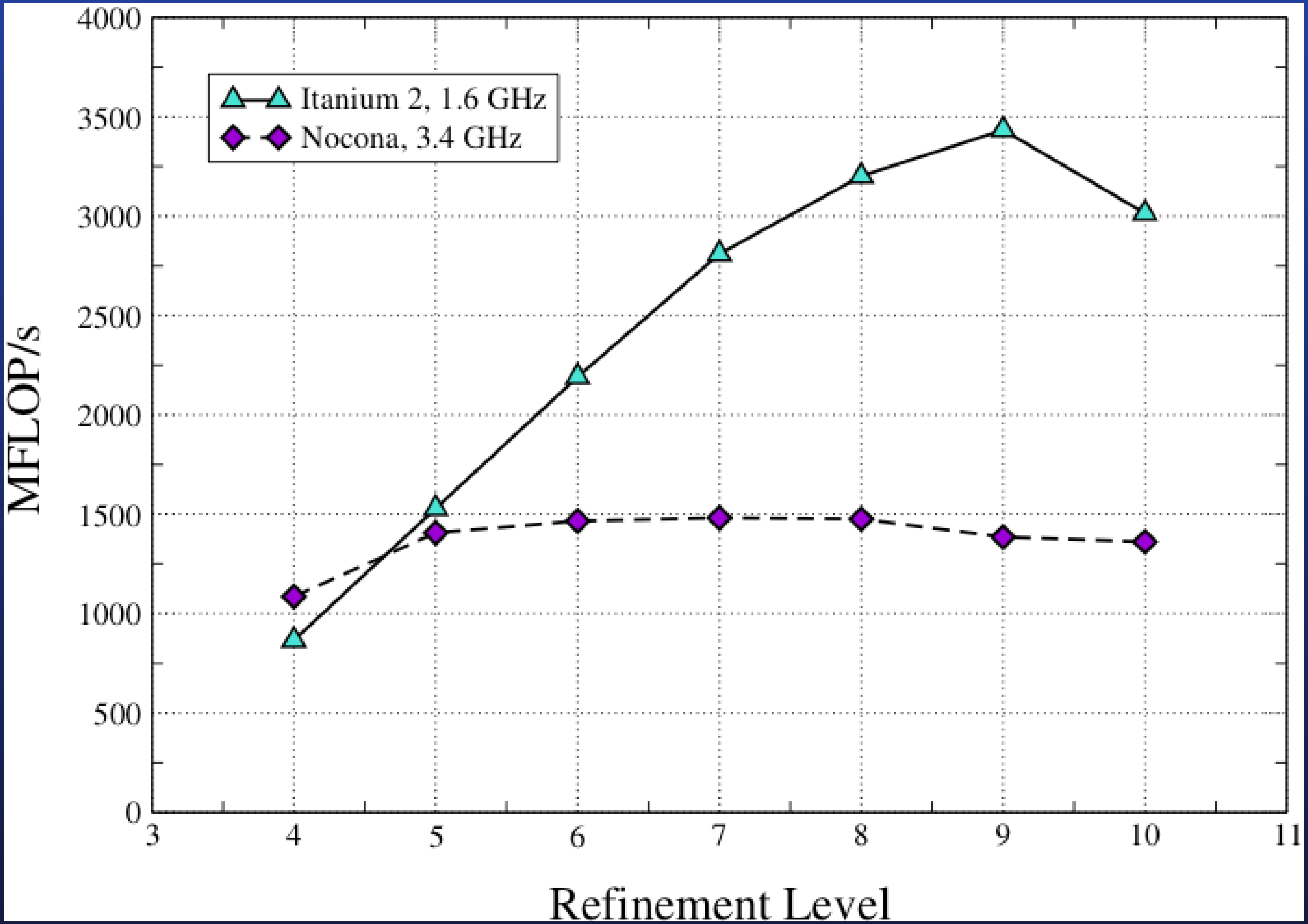
Structured Interior

# Refinement example



Structured Interior

# HHG: Serial efficiency



# HHG: Parallel Scalability

| #Procs | #DOFS x 10^6 | #Els x 10^6 | #Input Els | GFLOP/s | Time [s] |
|---|---|---|---|---|---|
| 64 | 2,144 | 12,884 | 6144 | 100/75 | 68 |
| 128 | 4,288 | 25,769 | 12288 | 200/147 | 69 |
| 256 | 8,577 | 51,539 | 24576 | 409/270 | 76 |
| 512 | 17,167 | 103,079 | 49152 | 762/545 | 75 |
| 1024 | 17,167 | 103,079 | 49152 | 1,456/964 | 43 |

Parallel scalability of Poisson problem:

Machine - SGI Altix (Itanium2 1.6 GHz)

To be published as: *Is $1.7\times 10^{10}$ unknowns the largest finite element system that can be solved today?* SC 2005.

# Summary

- Modern instruction set architectures put high demands on compilers
- Hardware aware low level optimizations can yield large improvements (e.g. a factor up to five) in performance. Can you afford to waste this potential?
- HHG data structures for finite elments lead to a scalable, parallel solver that achieves very good performance

# Thanks to ...

- DiME  (Data Local Iterative Methods) project funded since 1996 by DFG
  - http://www10.informatik.uni-erlangen.de/en/Research/Projects/DiME/
- KONWIHR
- Elite Network of Bavaria
- Thanks to
  - our collaborators: A. Bode, J. Weidendorfer, W. Karl, C. Weiß, C. Körner, H.J. Schmid, W. Peukert, G. Wellein, G. Hager, T. Zeiser, M. Thies, G. Greiner, P. Kipfer, S. Meinlschmid
  - graduate and undergraduate students: M. Kowarschik, M. Mohr,  J. Wilke, Harald Pfänder, G. Radzom,  N, Thürey, I. Christadler, S. Hausmann, K. Iglberger, C. Freundl, T. Pohl, U. Fabricius, S. Donath, V. Daum, J. Thies, M. Sonntag, A. Hausner, F. Fleißner, E. Lang, M. Zetlmeisl, S. Weigand
  - especially: Markus Stürmer (student project)

# The End: Real Time Flow Simulation

Zur Anzeige wird der QuickTime
Dekompressor
bentigt.