Performance Engineering for Multi- and Manycores: Unveiling some Mysteries of Application Performance

Georg Hager

Erlangen Regional Computing Center (RRZE)

University of Erlangen-Nuremberg, Germany

ISC12 Invited Session

"Application Performance: Lessons Learned from Petascale Computing"

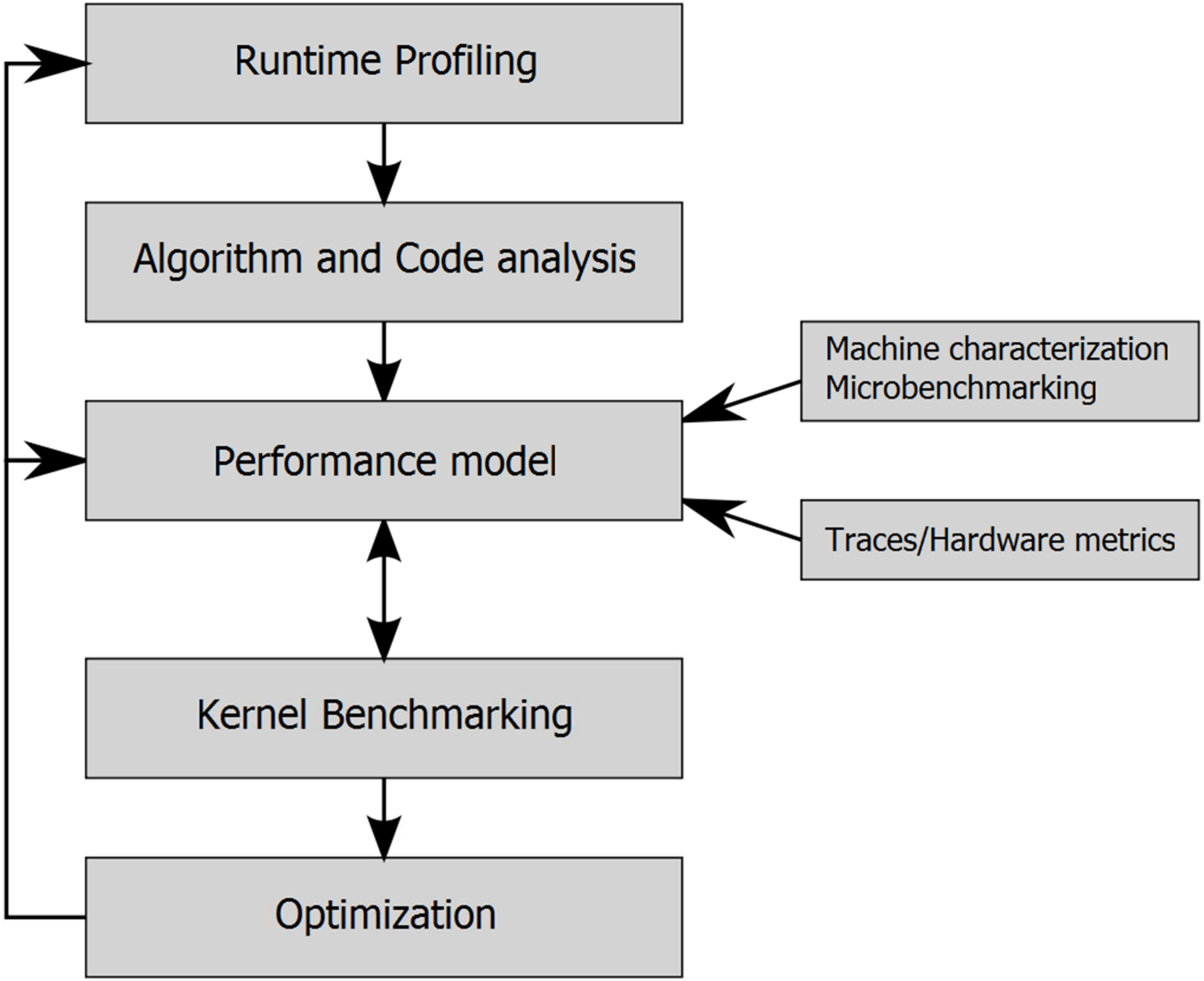
Hamburg, Germany, June 18, 2012

Statements

Performance data and code optimizations are useless except when put into the context of a suitable performance model!

Efficiency is made on the single core and chip level. Adding more hardware can only make it worse!

The Performance Engineering Cycle



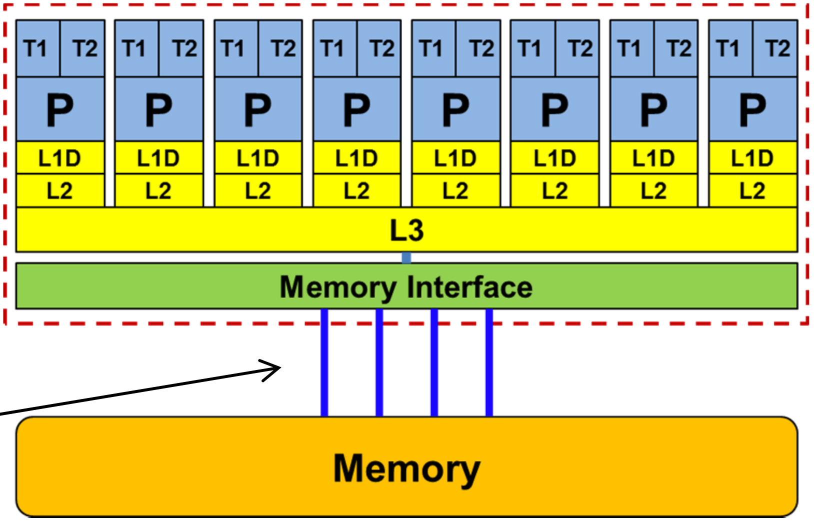
Example: Red-black Gauss-Seidel smoother

- **Simple iterative solver for boundary value problems**
- **Memory-bound for large data sets**

- **Benchmark platform:**

  One socket Intel Sandy Bridge EP
  2.7 GHz base frequency
  Up to 3.5 GHz Turbo Mode

  Memory bandwidth ≈ 36 Gbyte/s



- **Performance metric: Lattice site Updates per second (LUP/s)**
  - 1 LUP ↔ 48 bytes of memory traffic

A very simple example: Red-black Gauss-Seidel smoother

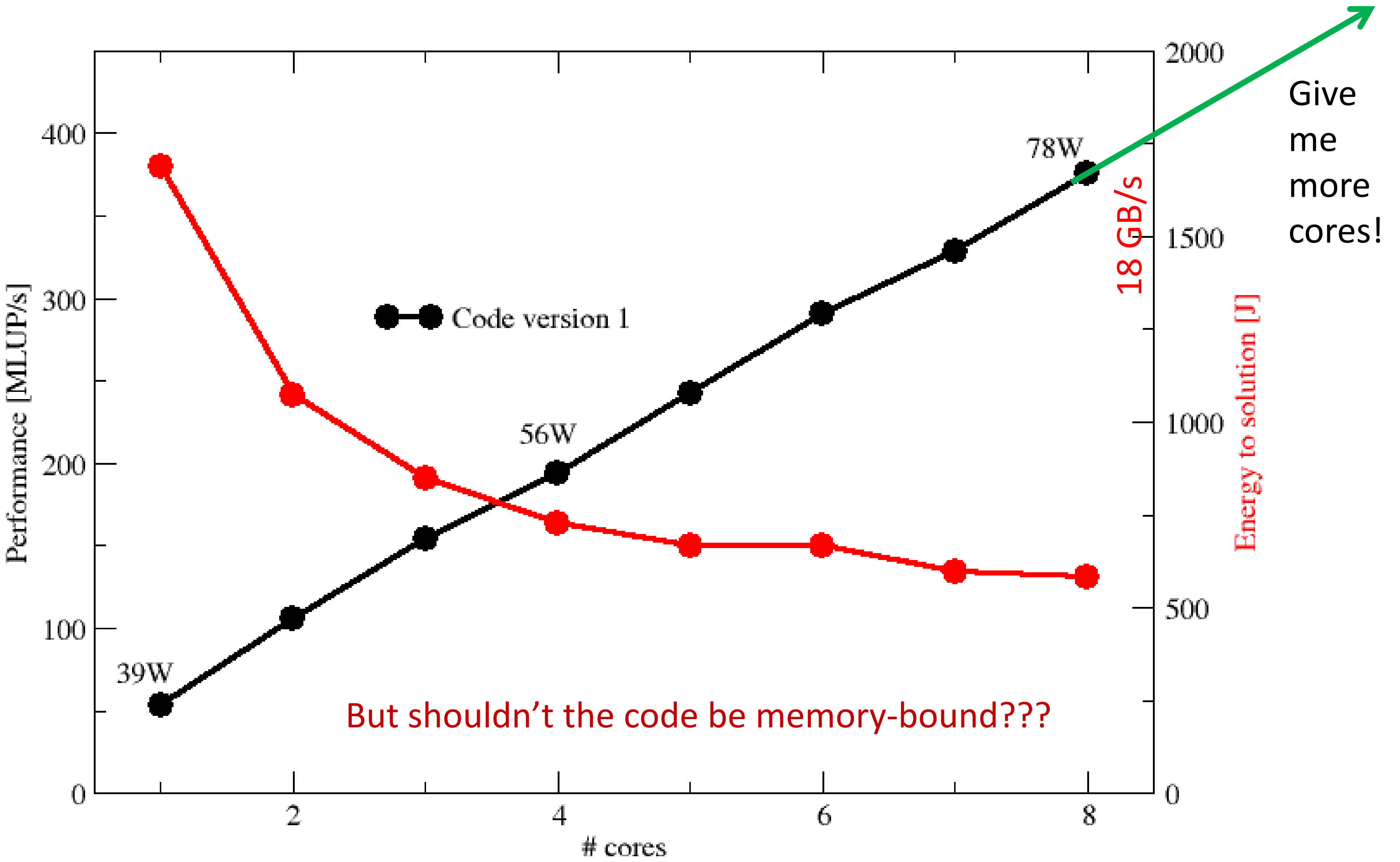
Code for one lattice site update (version 1)

```
template <typename T>
   inline T update(
       grid<T> const & u
     , grid<T> const & rhs
     , typename grid<T>::size_type x
     , typename grid<T>::size_type y
     , T hx_sq
     , T hy_sq
     , T div
     , T relaxation ) {
       return
           u(x, y)
           + (
               (
                   (
                       (u(x - 1, y) + u(x - 1, y)) / hx_sq
                     + (u(x, y - 1) + u(x, y + 1)) / hy_sq
                     + rhs(x, y)
                   )
                   / div
               )
               - u(x, y)
           ) * relaxation;
   }
```

```
// Main loop
for(iter=0; iter<MAX_ITER; iter++) {
  // red sweep
  for(y = 1; y < n_y-1; ++y)
    for(x=(y%2)+1; x<n_x-1; x+=2)
      u(x, y) = update(u, rhs, x, y,
                       hx_sq, hy_sq, div_,
                       relaxation);
  // black sweep
  for(y = 1; y < n_y-1; ++y)
    for(x=((y+1)%2)+1; x<n_x-1; x+=2)
      u(x, y) = update(u, rhs, x, y,
                       hx_sq, hy_sq, div_,
                       relaxation);
}
```

Standard five-point stencil

Code version 1: A scalable implementation



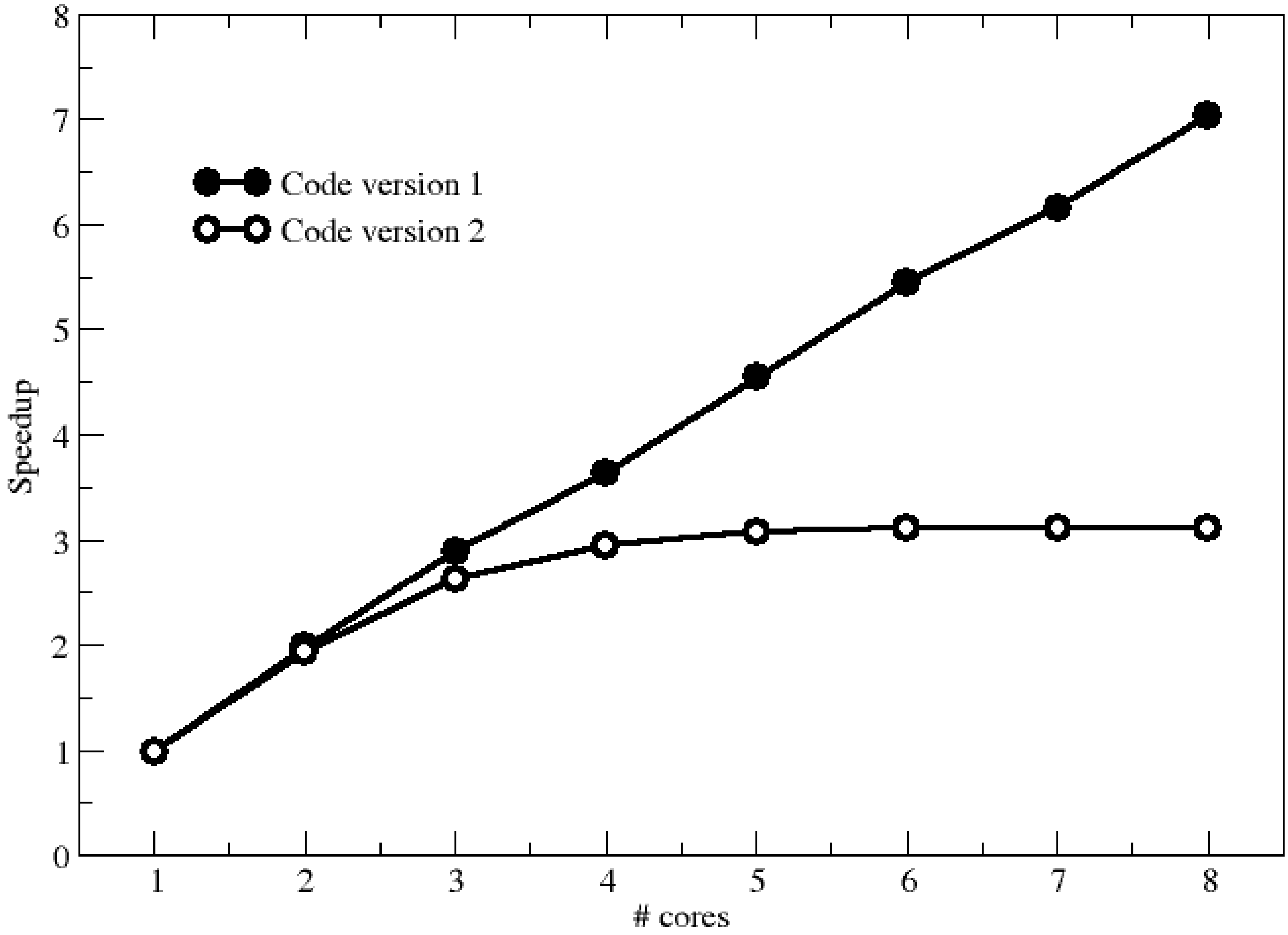
Example: Red-black Gauss-Seidel smoother

Code for one lattice site update (version 2)

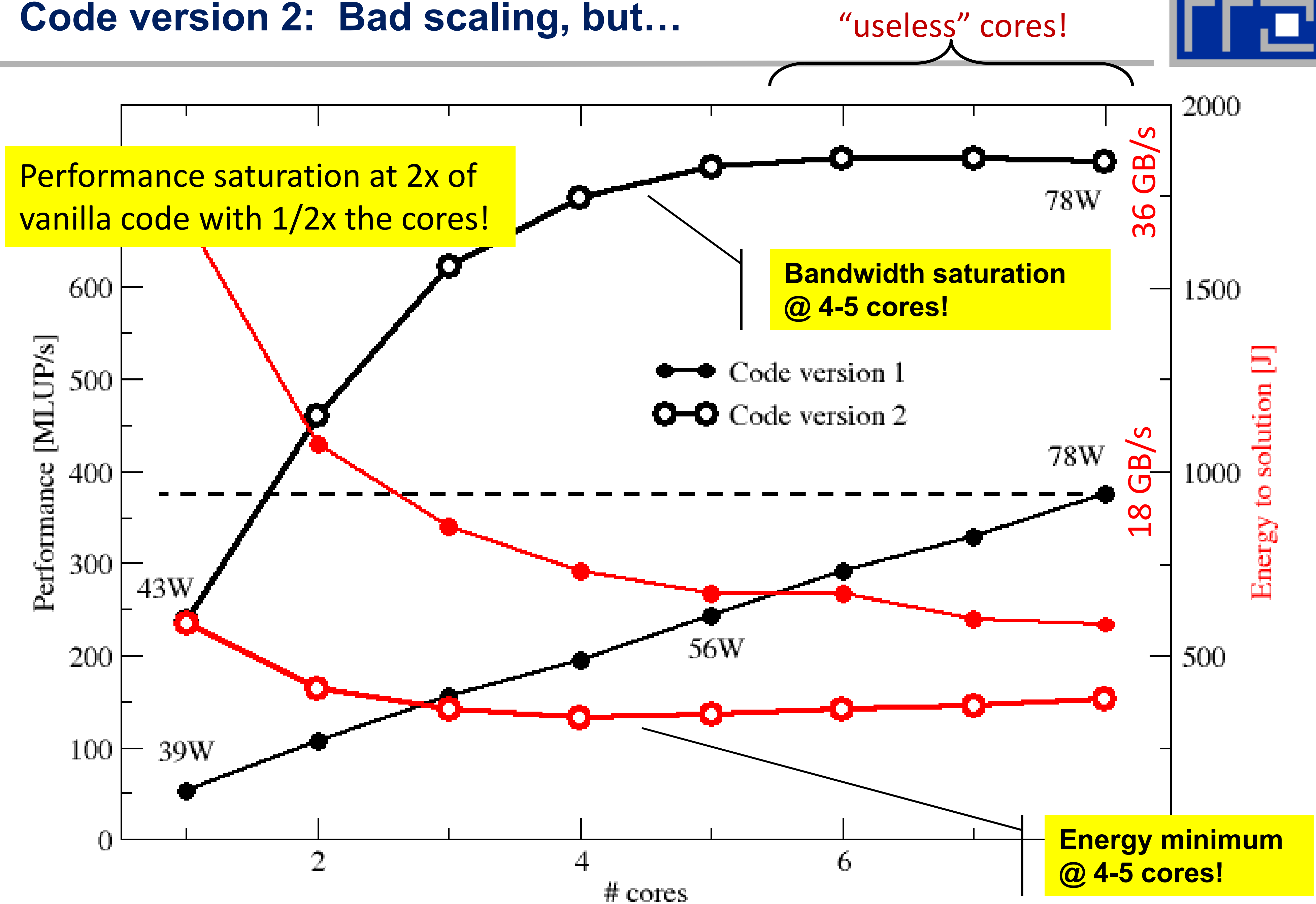
```
template <typename T>
   inline T update(
       grid<T> const & u
     , grid<T> const & rhs
     , typename grid<T>::size_type x
     , typename grid<T>::size_type y
     , T r_hx_sq
     , T r_hy_sq
     , T r_div
     , T relaxation ) {
       return
           u(x, y)
           + (
               (
                   (
                       (u(x - 1, y) + u(x - 1, y)) * r_hx_sq
                     + (u(x, y - 1) + u(x, y + 1)) * r_hy_sq
                     + rhs(x, y)
                   )
                   * r_div
               )
               - u(x, y)
           ) * relaxation;
   }
```

Replaced divides by MULTs

Scalability for two different code versions



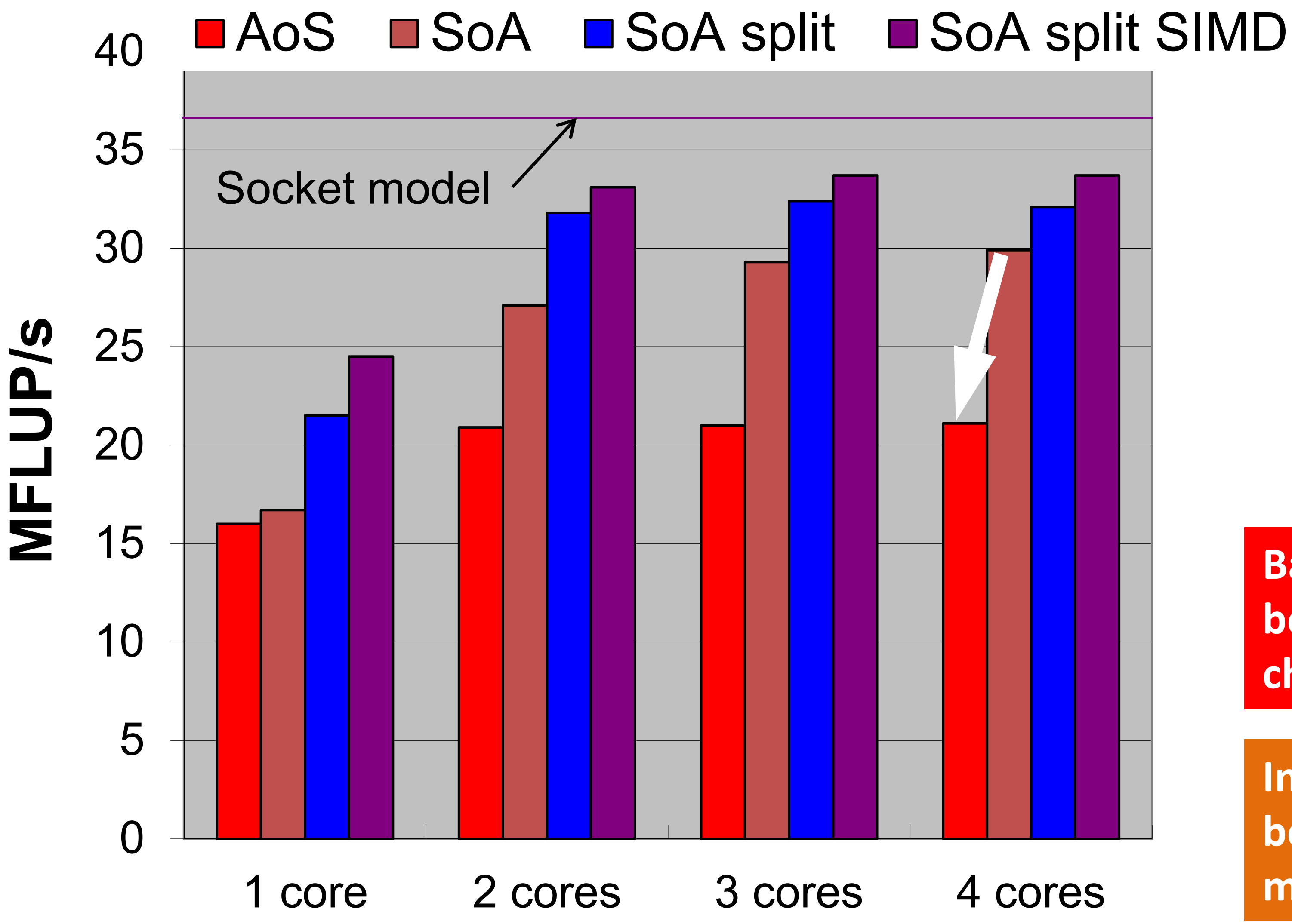
Code version 2: Bad scaling, but…

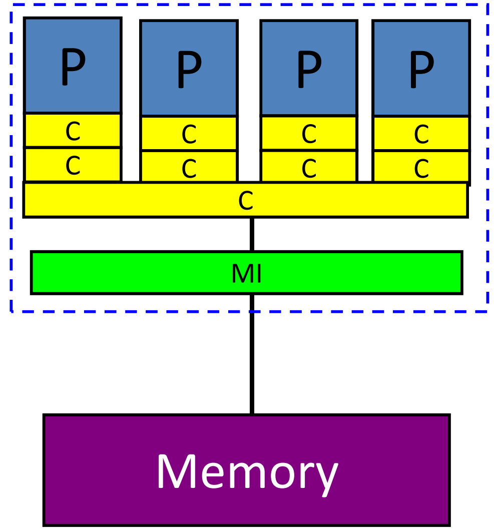


Can we “heal” bad single-core performance by using more cores on the chip?

Healing bad single-core performance: Lattice-Boltzmann solver on Intel Sandy Bridge

Benchmark: Double precision, lid-driven cavity with 230^3 fluid cells



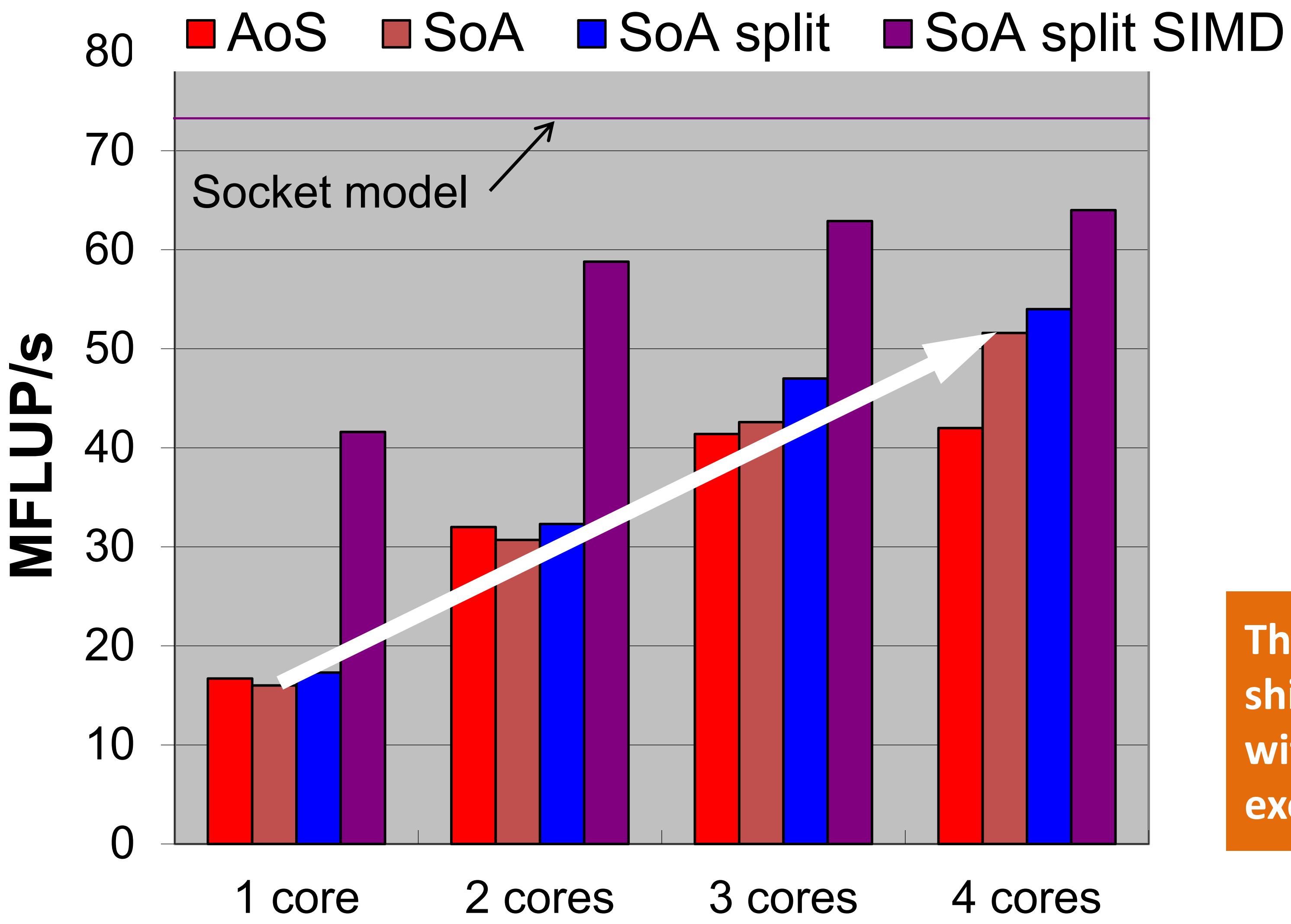


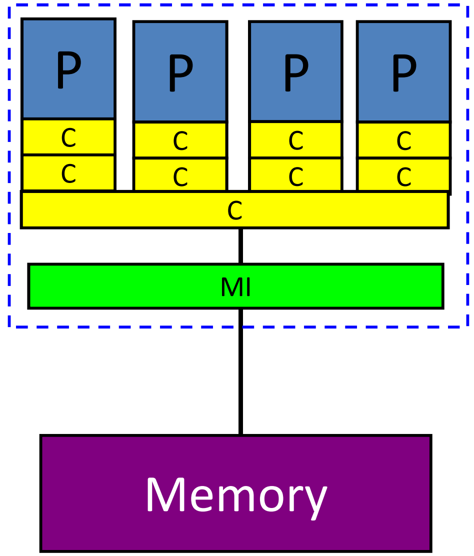
Bad data layout cannot be compensated by on-chip parallelism!

In-core inefficiency can be "healed" by using more cores!

Healing bad single-core performance: Lattice-Boltzmann solver on Intel Sandy Bridge

Benchmark: Single precision, lid-driven cavity with 230^3 fluid cells

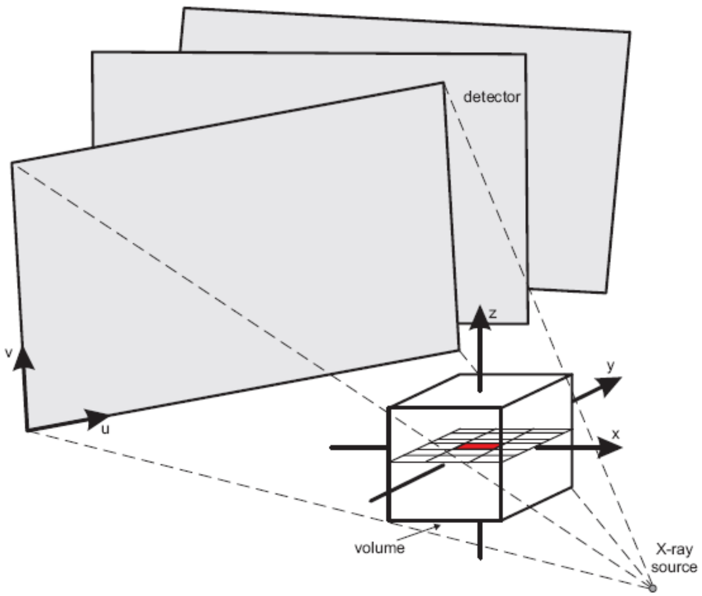




The "healing point" shifts to more cores with non-SIMD execution!

Does it stop at the roofline model?

A more complex example:
A medical image reconstruction code on Sandy Bridge

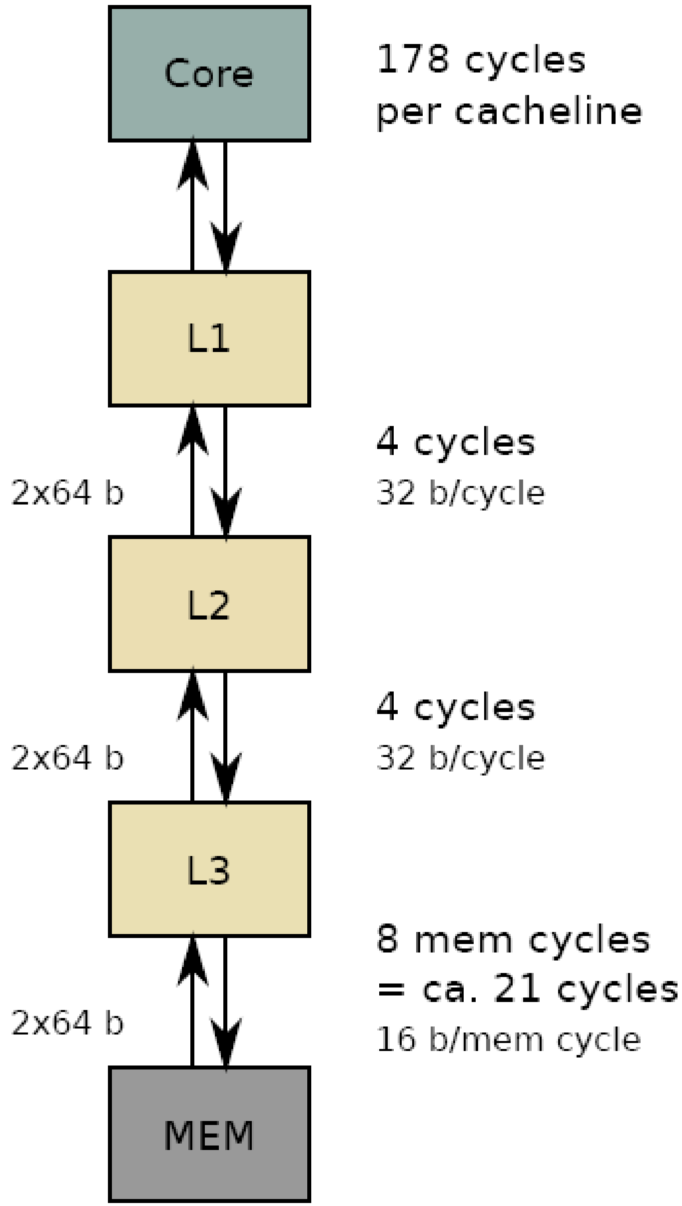


Sandy Bridge EP (8 cores, 2.7 GHz base freq.)

Test case	Runtime [s]	Power [W]	Energy [J]
8 cores, plain C	**90.43**	90	8110
8 cores, SSE	29.63	93	2750
8 cores (SMT), SSE	22.61	102	2300
8 cores (SMT), AVX	**18.42**	111	2040

Faster code → less energy

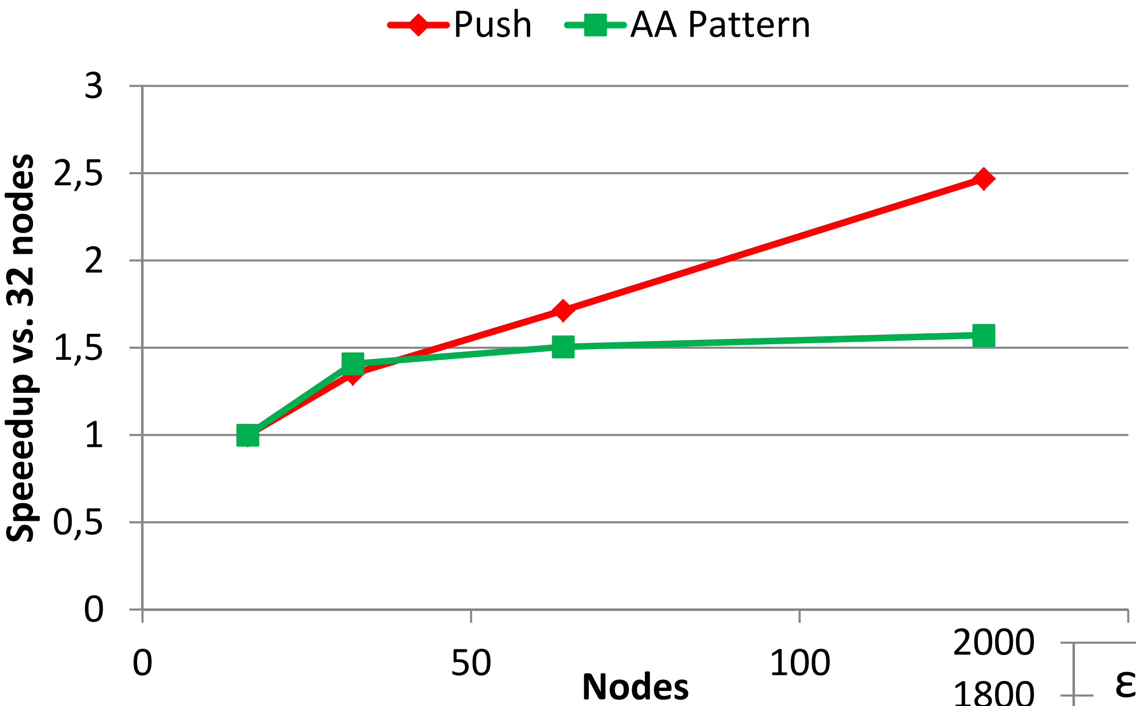
- **Runtime analysis: Backprojection loop dominates**
- **First shot: Roofline model predicts memory-bound situation**
- **HPM measurement: Memory Bandwidth not saturated**
- **Refined performance model**
  - Core execution
  - Cache line transfer within the cache hierarchy
  - Cache line transfer to/from memory

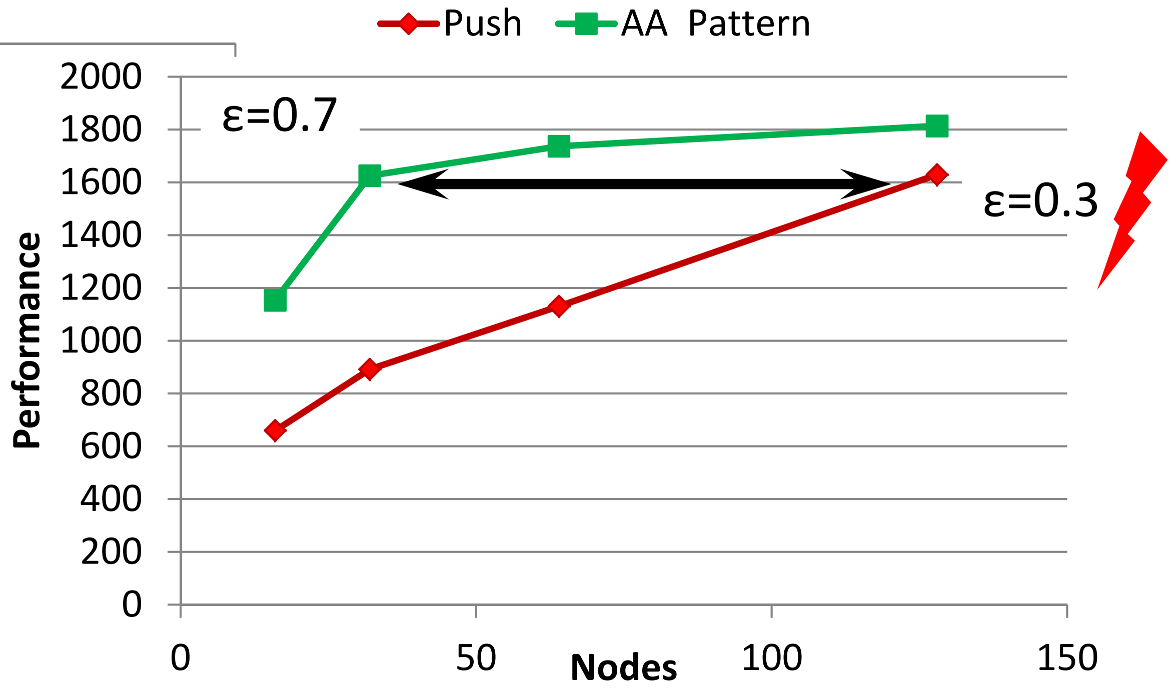


- **Results**
  - Parallel execution far from memory-bound
  - Core execution dominates
  - Model prediction within 12-20% of actual performance

How about healing bad node-level performance by using more nodes?

Scaling up a lattice-Boltzmann solver





4x more science per compute cycle (at scale) and >2x better parallel efficiency by single-node optimization!

Thank You.

Bundesministerium
für Bildung
und Forschung

www.skalb.de

SKALB (01 IH08003A)

KONWIHR II

OMI4papps

Jan Treibig, Markus Wittmann, Johannes Habich, Gerhard Wellein