Performance Portable Tracking of Evolving Surfaces

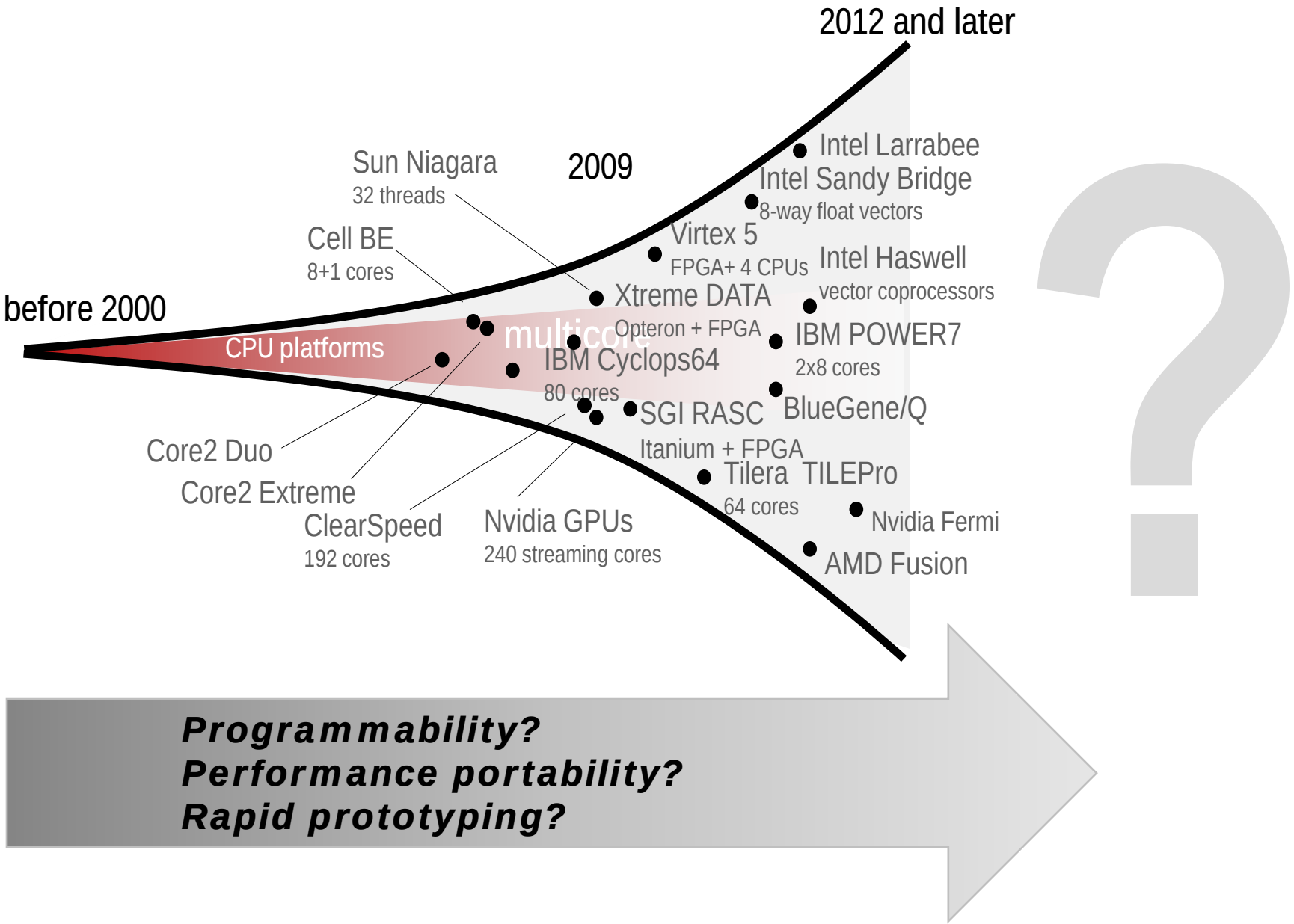
Wei Yu
Citadel Investment Group

Franz Franchetti, James C. Hoe, José M. F. Moura
Carnegie Mellon University

Tsuhan Chen
Cornell University

This work was supported by DARPA, ONR, NSF, Intel, Mercury, and ITRI

The Future is Parallel and Heterogeneous



Software's Slow Pace of Change

Popular performance programming languages

- **1953:** Fortran
- **1973:** C
- **1985:** C++
- **1997:** OpenMP
- **2007:** CUDA

Popular performance libraries

- **1979:** BLAS
- **1992:** LAPACK
- **1994:** MPI
- **1995:** ScaLAPACK
- **1995:** PETSc
- **1997:** FFTW

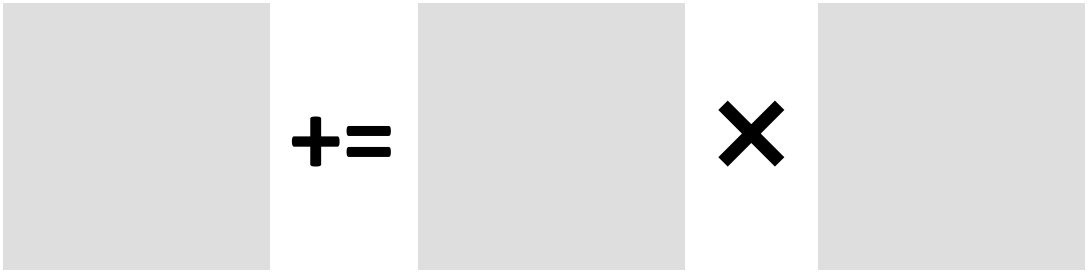
Popular productivity/scripting languages

- **1987:** Perl
- **1989:** Python
- **1993:** Ruby
- **1995:** Java
- **2000:** C#

Compilers: The Fundamental Problem

Matrix-matrix multiplication

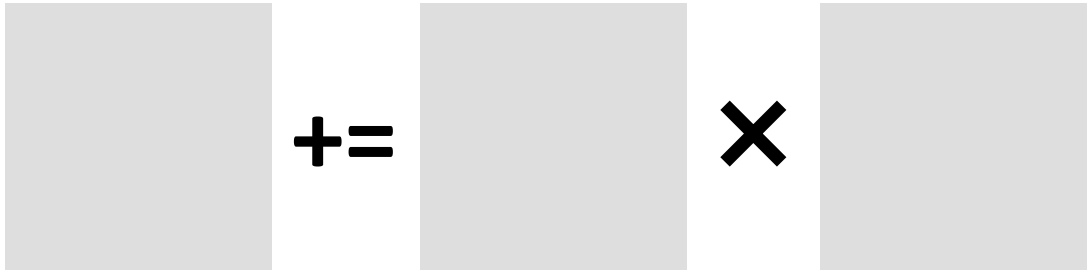
```
for i=1:N
  for j=1:N
    for k=1:N
      C[i,j] += A[i,k]*B[k,j]
```



Compilers: The Fundamental Problem

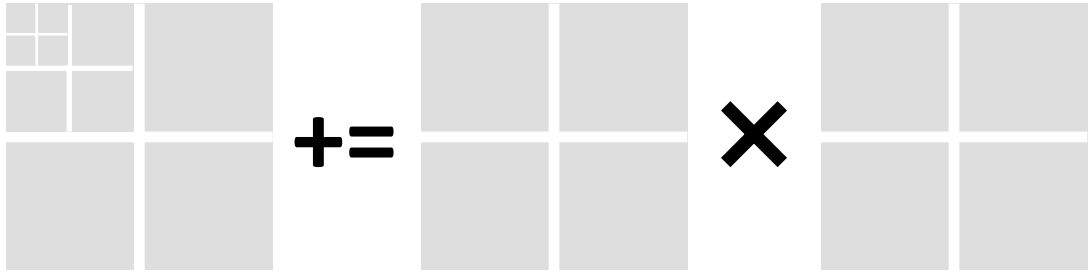
Matrix-matrix multiplication

```
for i=1:N
  for j=1:N
    for k=1:N
      C[i,j] += A[i,k]*B[k,j]
```



Tiled+unrolled+parallelized+SIMDized+prefetching matrix-matrix multiplication

```
#pragma omp parallel for
for i=1:NB:N
  for j=1:NB:N
    for k=1:NB:N
      for i0=i:NU:i+NB
        #pragma prefetch B
        for j0=j:NU:j+NB
          for k0=k:NU:k+NB
            #pragma unroll
            for k00=k0:1:k0+NU
              for j00=j0:1:j0+NU
                #pragma vector always
                for i00=i0:1:i0+NU
                  C[i00,j00] += A[i00,k00]*B[k00,j00];
```



Problem: which transformation order? What parameter values?

Autotuning and Program Synthesis

Synthesis from Domain Math

- **Spiral**
  Signal and image processing, SDR
- **Tensor Contraction Engine**
  Quantum Chemistry Code Synthesizer
- **FLAME**
  Numerical linear algebra (LAPACK)

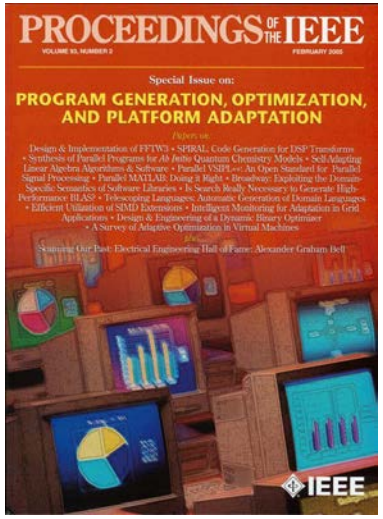
Compiler-Based Autotuning

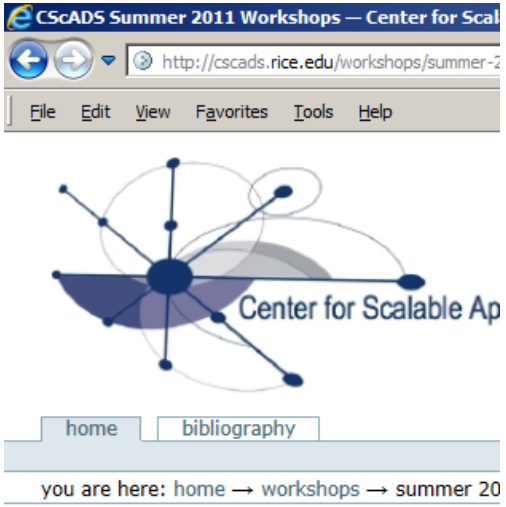
- **Polyhedral framework**
  IBM XL, Pluto, CHiLL
- **Transformation prescription**
  CHiLL, POET
- **Profile guided optimization**
  Intel C, IBM XL

Autotuning Numerical Libraries

- **ATLAS**
  BLAS generator
- **FFTW**
  kernel generator
- **Vendor math libraries**
  Code generation scripts

Autotuning Primer

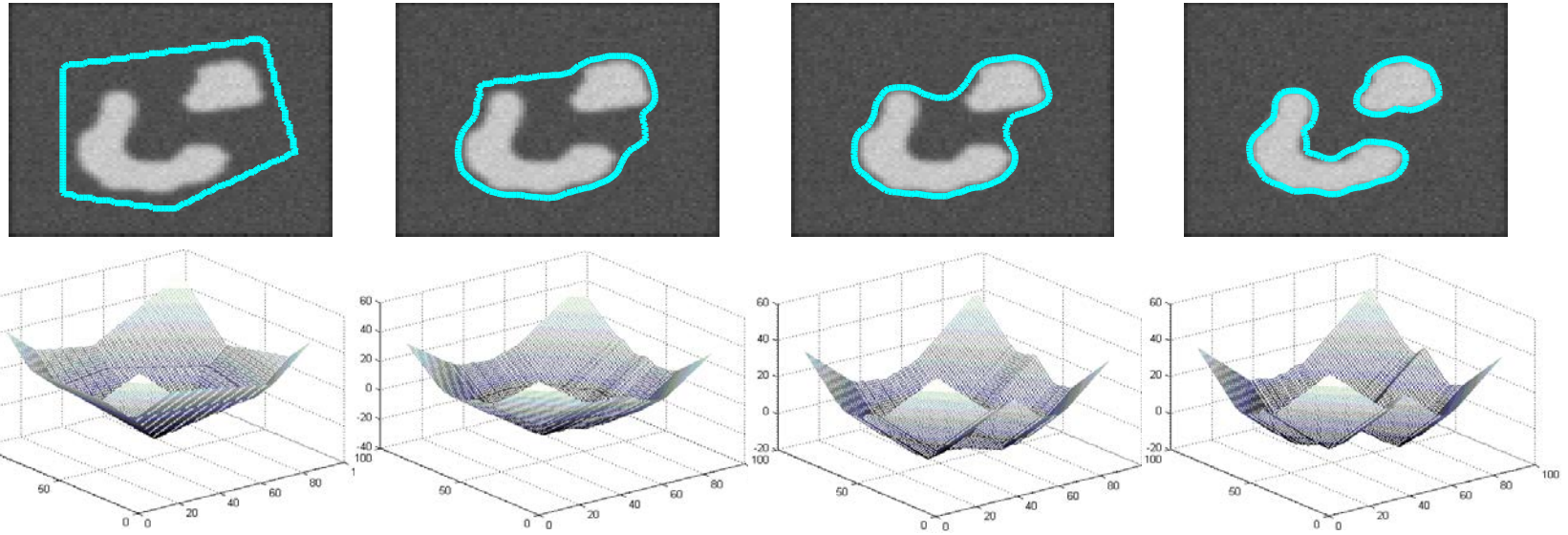




Level-Set Image Segmentation

Level Set Algorithm

- PDE-based image segmentation method
- Image produces force field on moving contour
- Embeds contour into n+1 dimensional level-set function



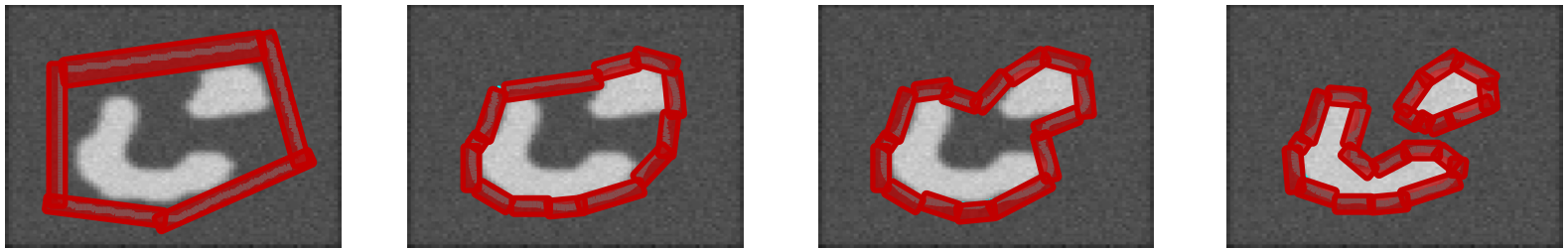
Level-Set Image Segmentation

Level Set Algorithm

- PDE-based image segmentation method
- Image produces force field on moving contour
- Embeds contour into n+1 dimensional level-set function

Narrow Band Level Set Algorithm

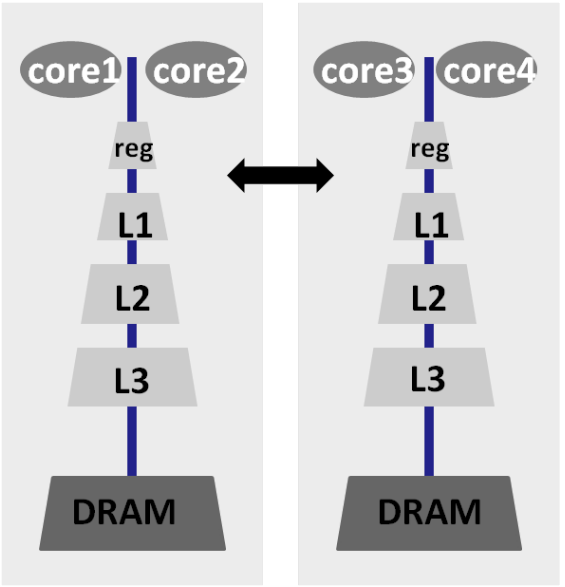
- Compute only in neighborhood of boundary
- Makes algorithm computationally tractable but complicated

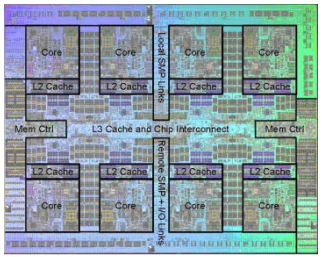


Target: Cache-Based Multicores

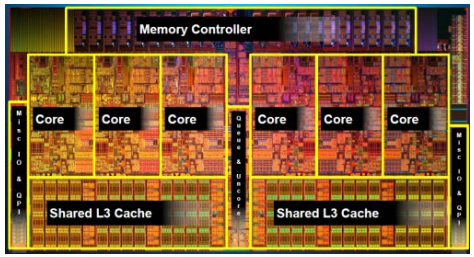
COTS Multicore

- Shared memory (SMP or NUMA)
- Cache-based memory (L1, L2, L3)
- SIMD vector instructions (SSE, AltiVec,…)

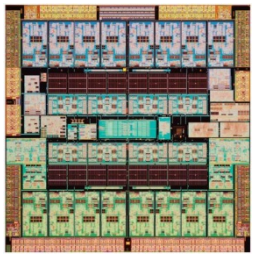




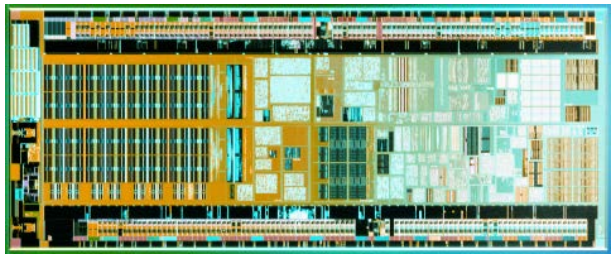
IBM POWER7
8 cores, 4-way SMT



Intel Core i7
8 cores, 2-way SMT



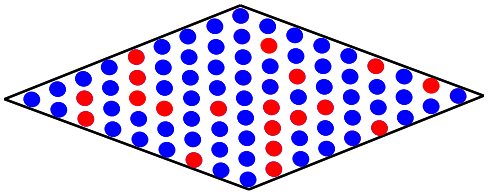
UltarSPARC T3
16 cores, 8-way SMT



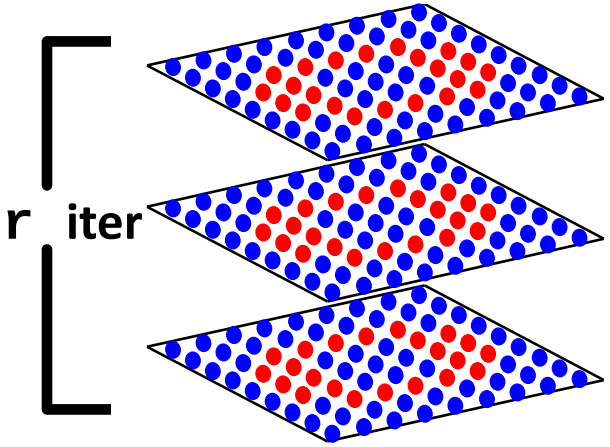
Intel Atom
1 cores, 2-way SMT

Why A Challenging Problem?

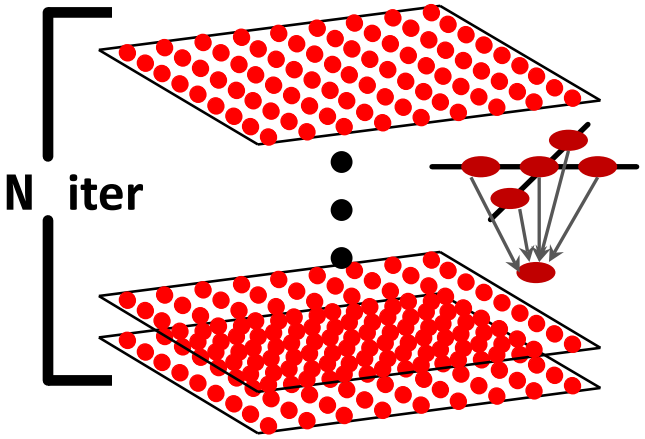
SparseMV



Narrow Band Level Set



Stencil



Memory bound

compute bound

O(1)

O(N)

sparse & irregular data structure	sparse data with dense substructure	dense & regular data structure

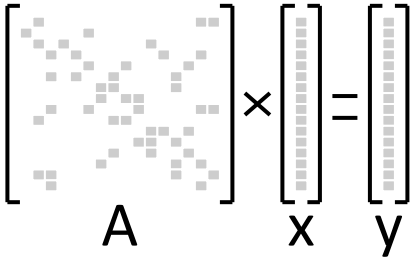
Approach

- **Understand algorithmic trade-offs**
  - Cheaper iterations vs. fewer iterations
  - Computation load vs. algorithm regularity
- **Understand the target hardware**
  - What are the costs and trade-offs?
  - What are good implementation choices and tricks?
- **Develop optimized parameterized implementation**
  - Parameters should expose machine/algorithm interaction
  - Parameterization may require macros or program generation
- **Autotune the algorithm and implementation parameters**
  - Parameter choice is unclear because of algorithm/architecture interaction
  - Autotuning is the last step in aggressive performance tuning

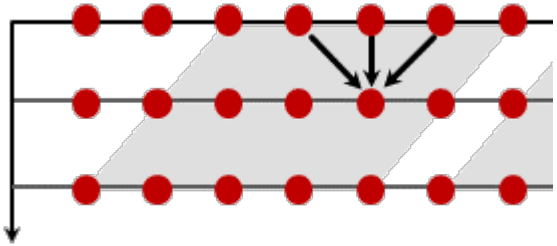
100x gain: *Optimized parameterized program: 50x, autotuning: 2x*

How to Get Level Set Up To Speed?

Sparse Linear Algebra



Time Skewing

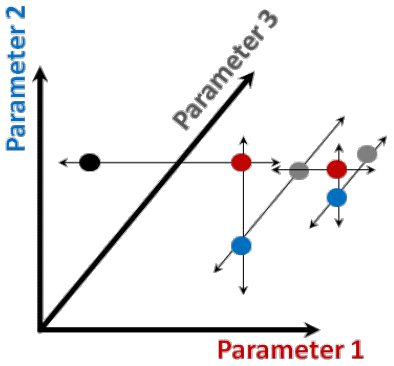


Code generation

```
for (int j=lj; j<=hj; j++)
  for (int i=li; i<=hi; i++)
    BI[j][i]=1;
```

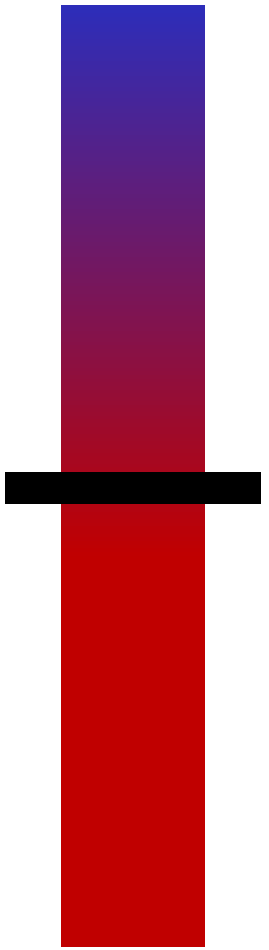
```
switch (case) {
  case (…):
    BI[j-1][i-1]=1;
    …
    BI[j+1][i+1]=1;
    break;
  case (…):
    …
}
```

Auto-tuning

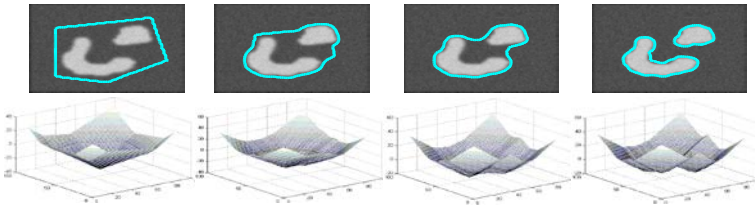


How Far Did We Go?

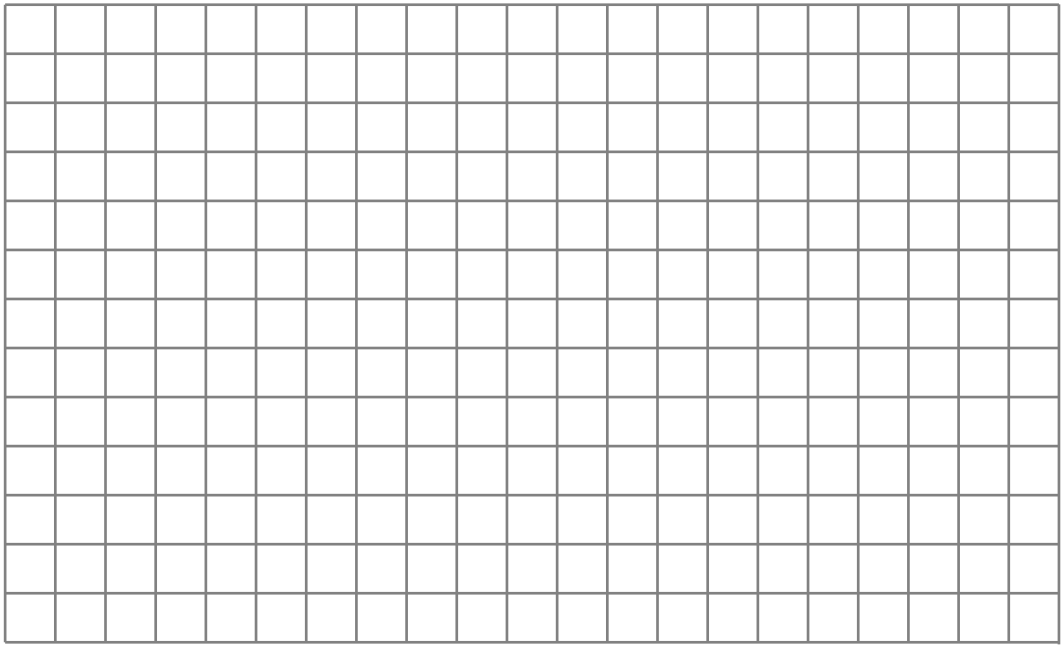
- **Level 0: simple C program**
  implements the algorithm cleanly
- **Level 1: C macros plus autotuning search script**
  use C preprocessor for meta-programming
- **Level 2: autotuning + scripting for code specialization**
  text-based program generation, specialization
- **Level 3: add compiler technology, synthesize parts**
  internal code representation, standard compiler passes
- **Level 4: synthesize the whole program from scratch**
  high level representation, domain-specific synthesis



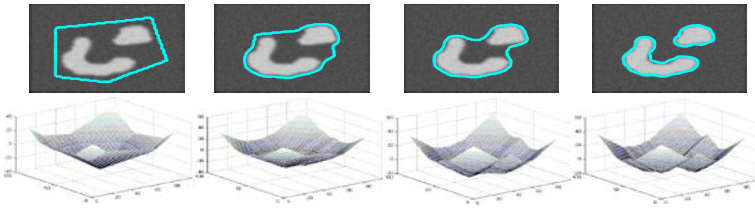
Narrow Band Level Set in a Nutshell



$$\phi^{t+1} = \Gamma(\phi^t(x \pm \Delta_x, y \pm \Delta_y))$$

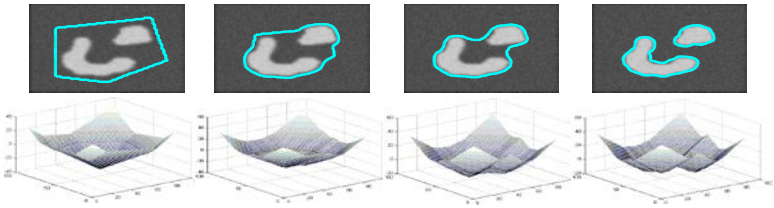


Narrow Band Level Set in a Nutshell

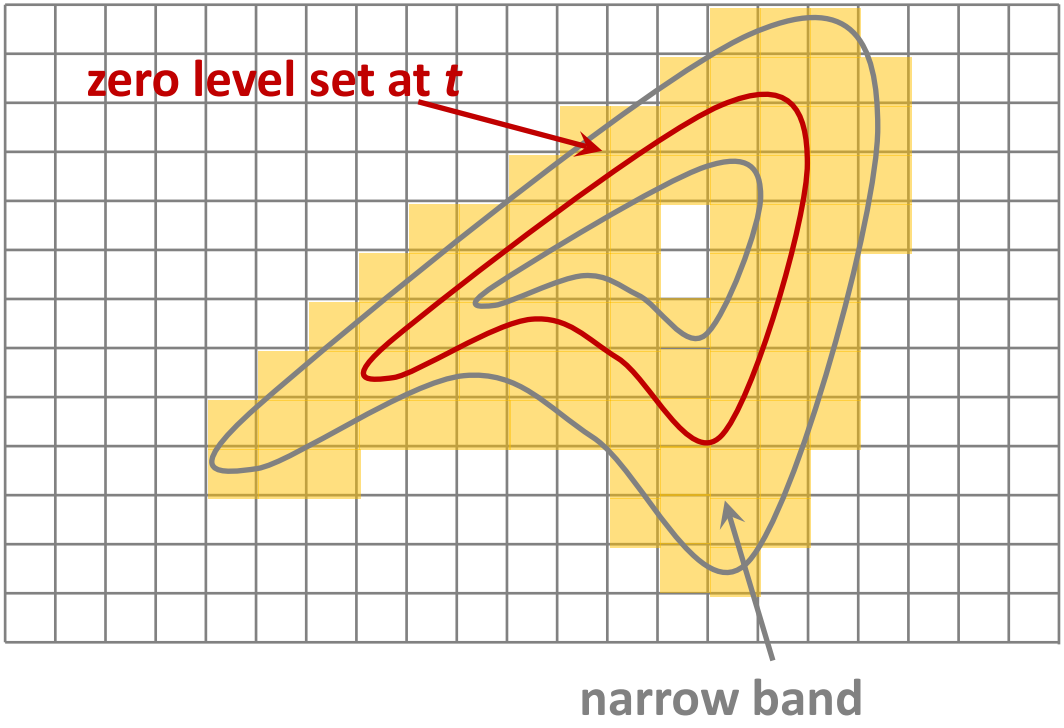


$$\phi^{t+1} = \Gamma(\phi^t(x \pm \Delta_x, y \pm \Delta_y))$$

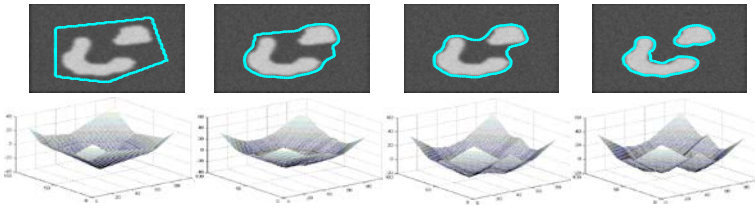
Narrow Band Level Set in a Nutshell



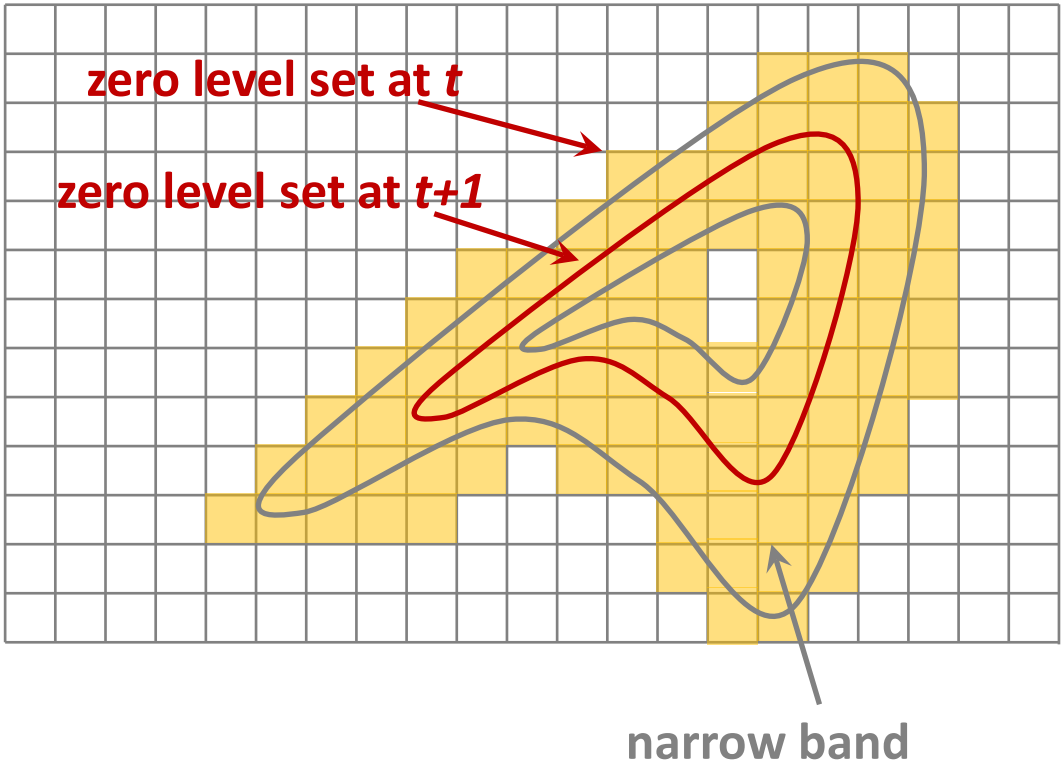
$$\phi^{t+1} = \Gamma(\phi^t(x \pm \Delta_x, y \pm \Delta_y))$$



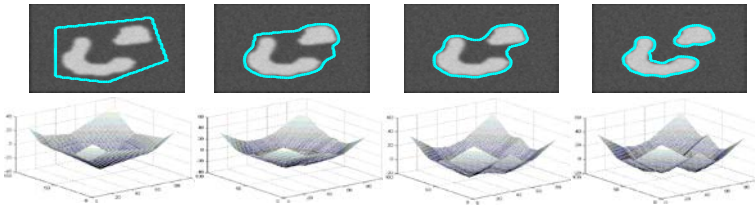
Narrow Band Level Set in a Nutshell



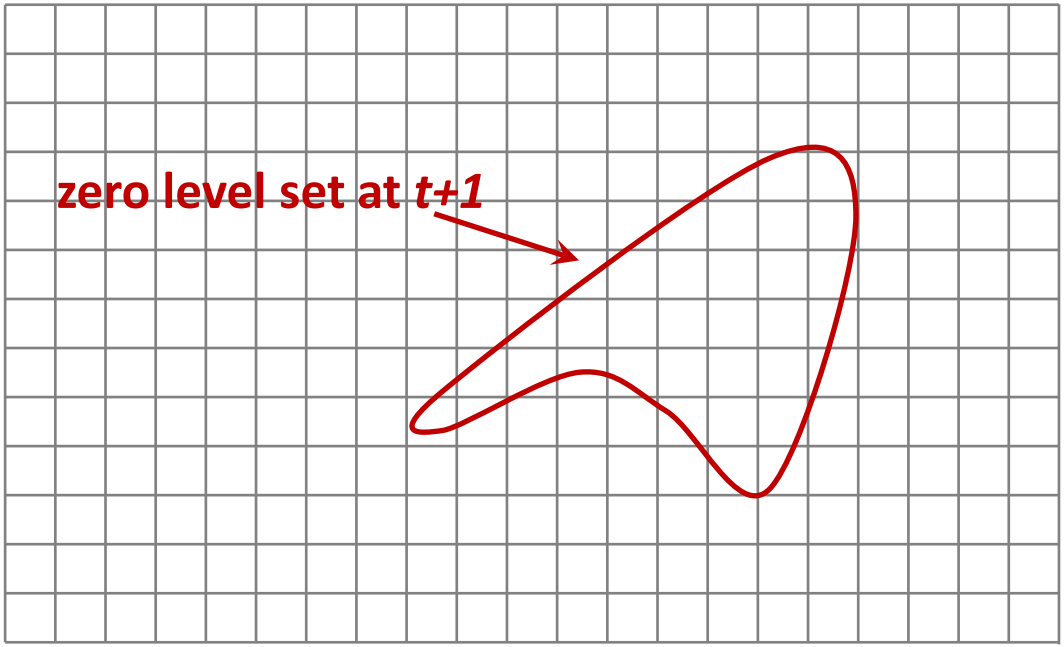
$$\phi^{t+1} = \Gamma(\phi^t(x \pm \Delta_x, y \pm \Delta_y))$$



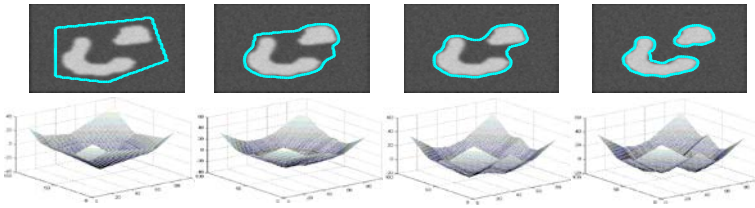
Narrow Band Level Set in a Nutshell



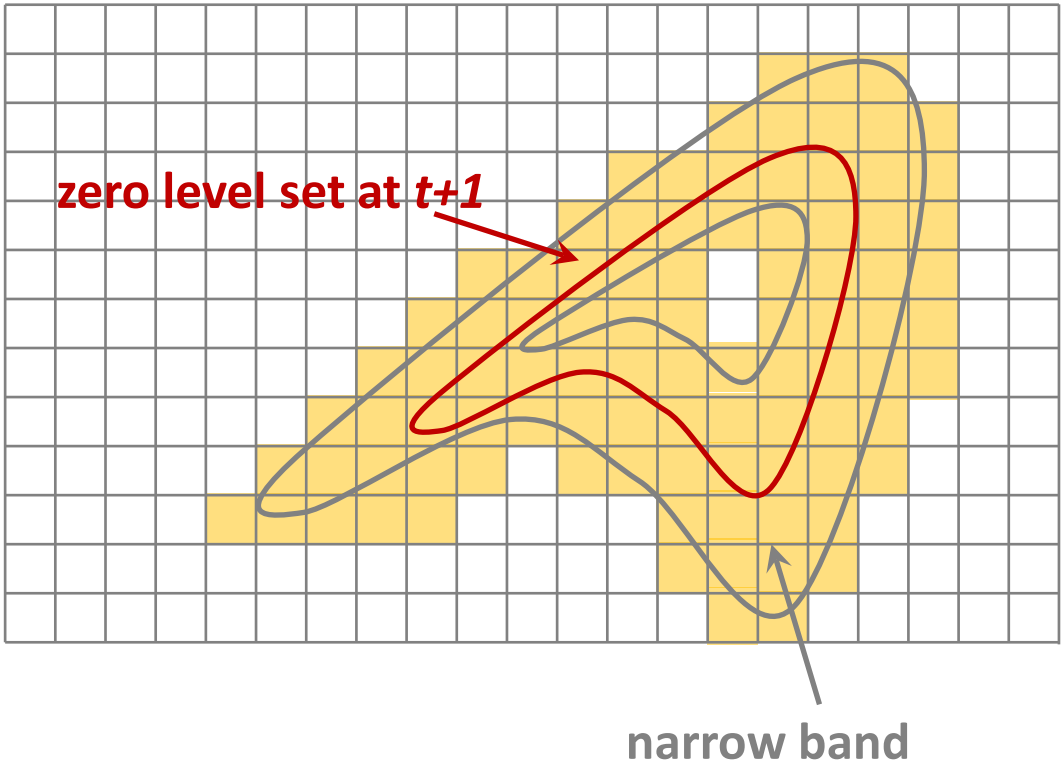
$$\phi^{t+1} = \Gamma(\phi^t(x \pm \Delta_x, y \pm \Delta_y))$$



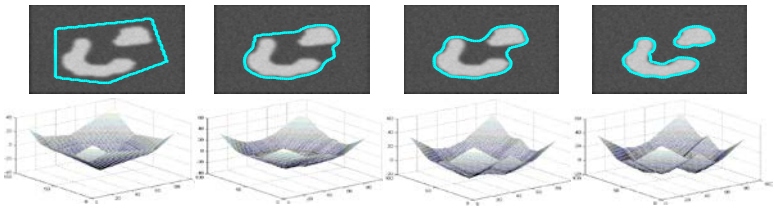
Narrow Band Level Set in a Nutshell



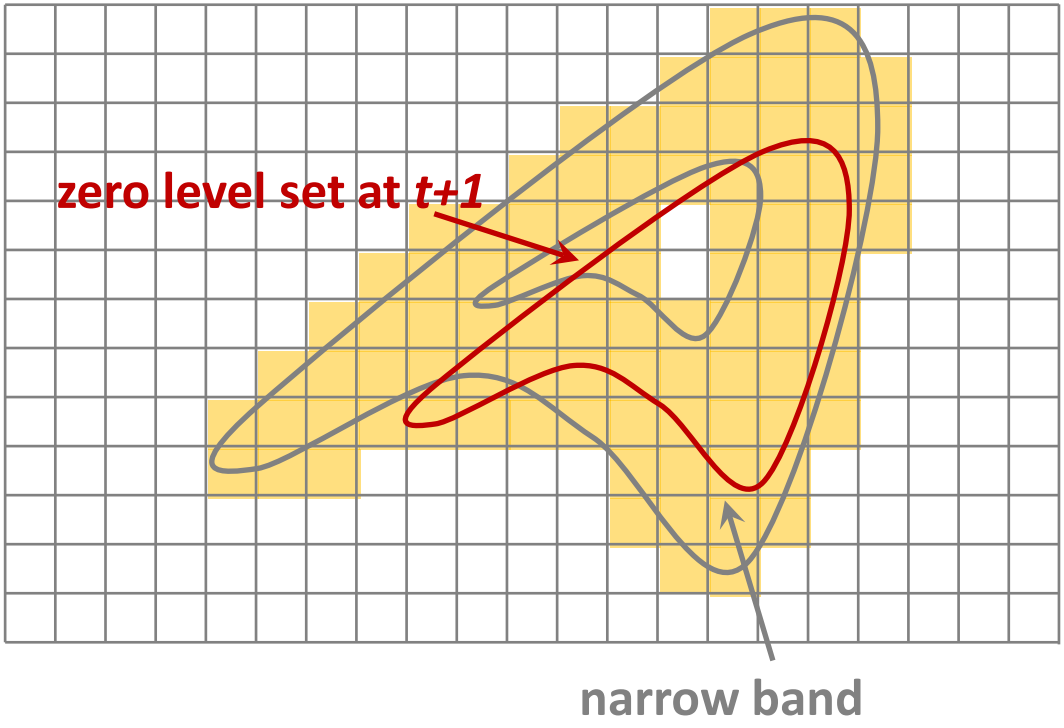
$$\phi^{t+1} = \Gamma(\phi^t(x \pm \Delta_x, y \pm \Delta_y))$$

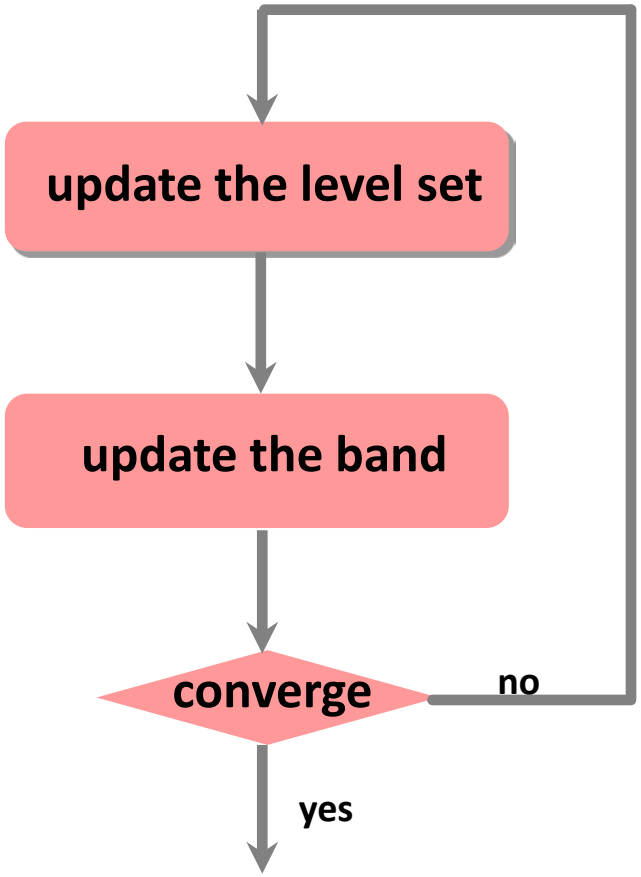


Narrow Band Level Set in a Nutshell

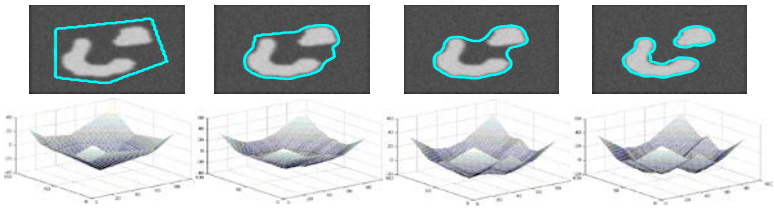


$$\phi^{t+1} = \Gamma(\phi^t(x \pm \Delta_x, y \pm \Delta_y))$$

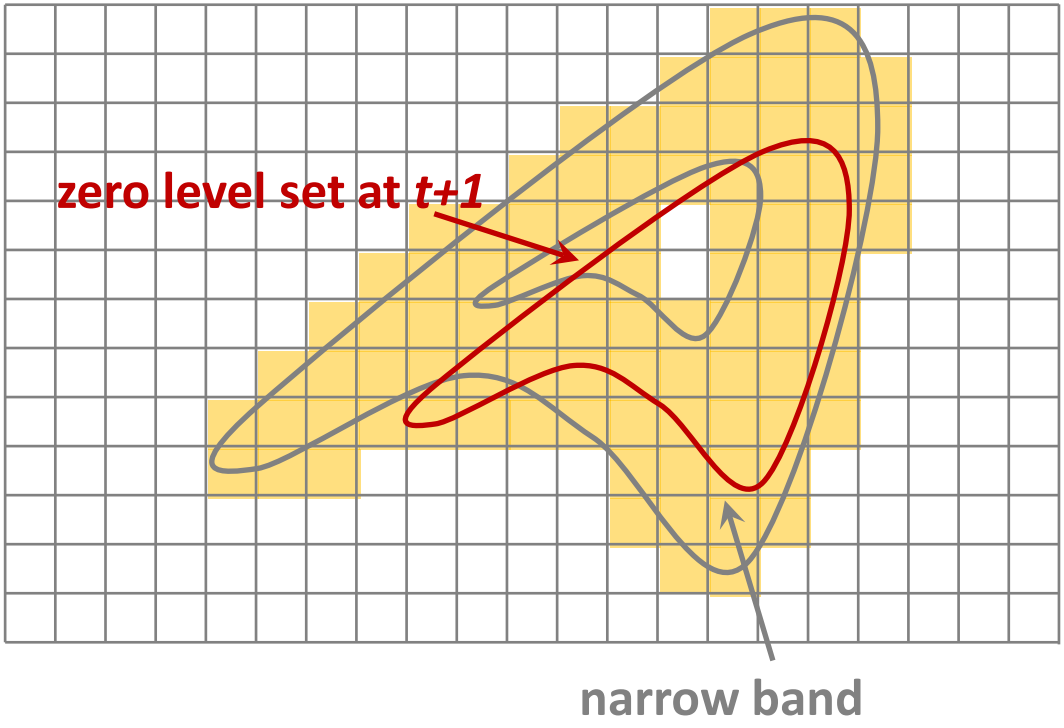


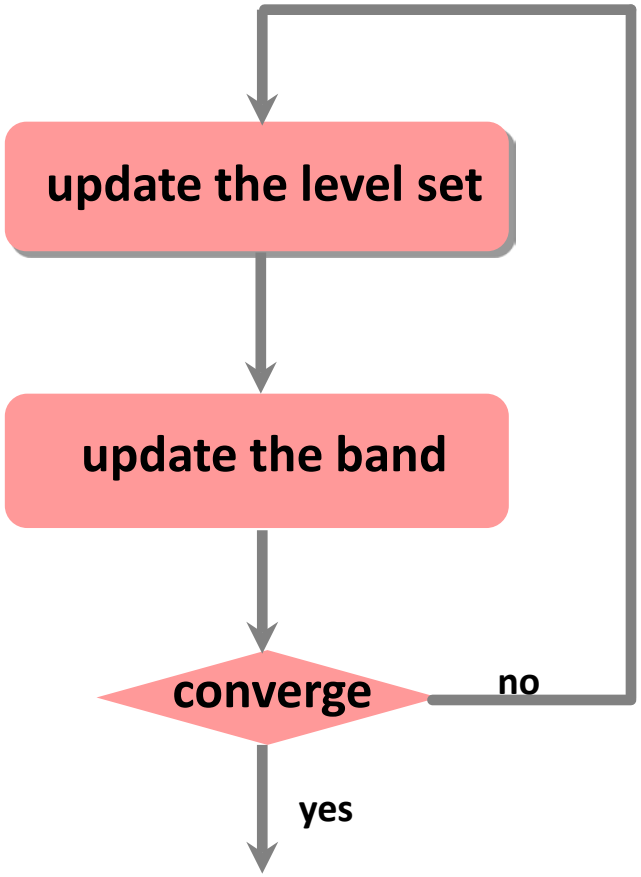


Narrow Band Level Set in a Nutshell



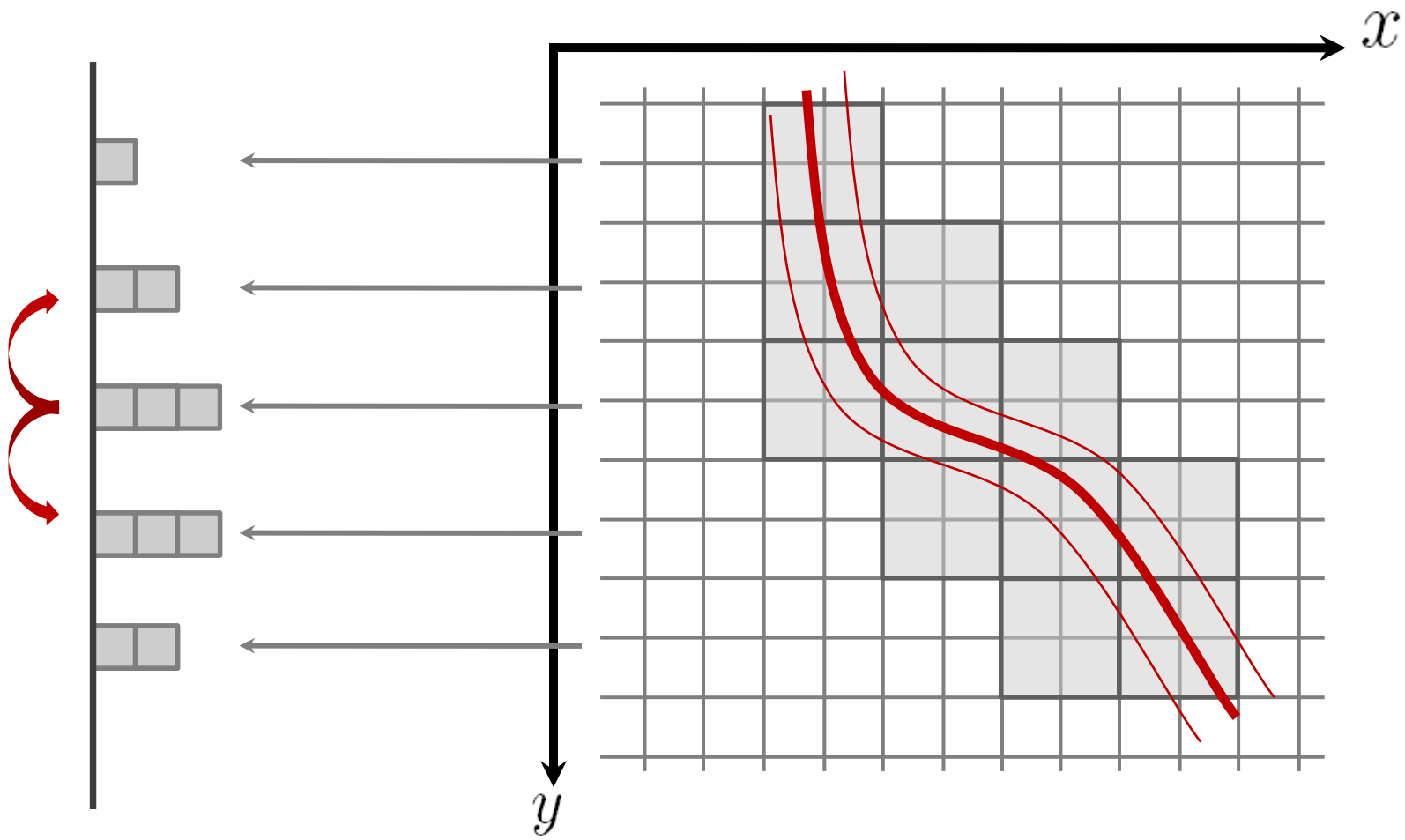
$$\phi^{t+1} = \Gamma(\phi^t(x \pm \Delta_x, y \pm \Delta_y))$$

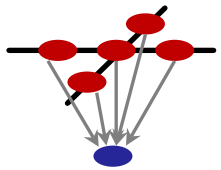




Autotuning parameter: narrow band width

Projection of Manifold into Dense Stencil

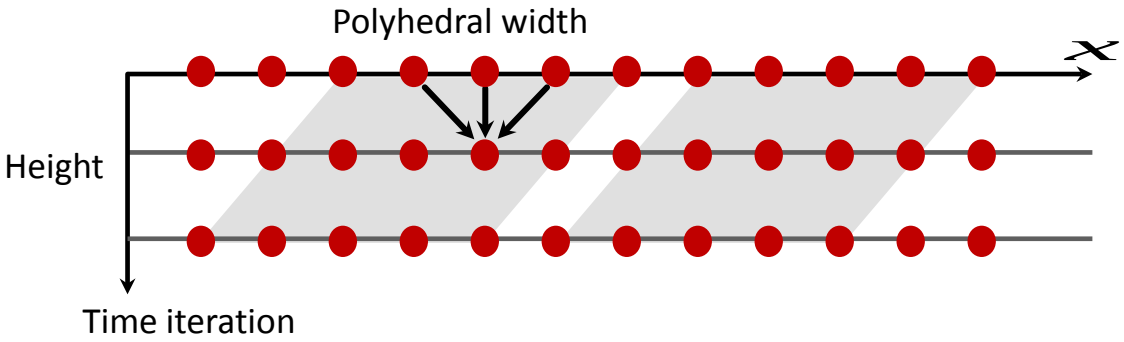




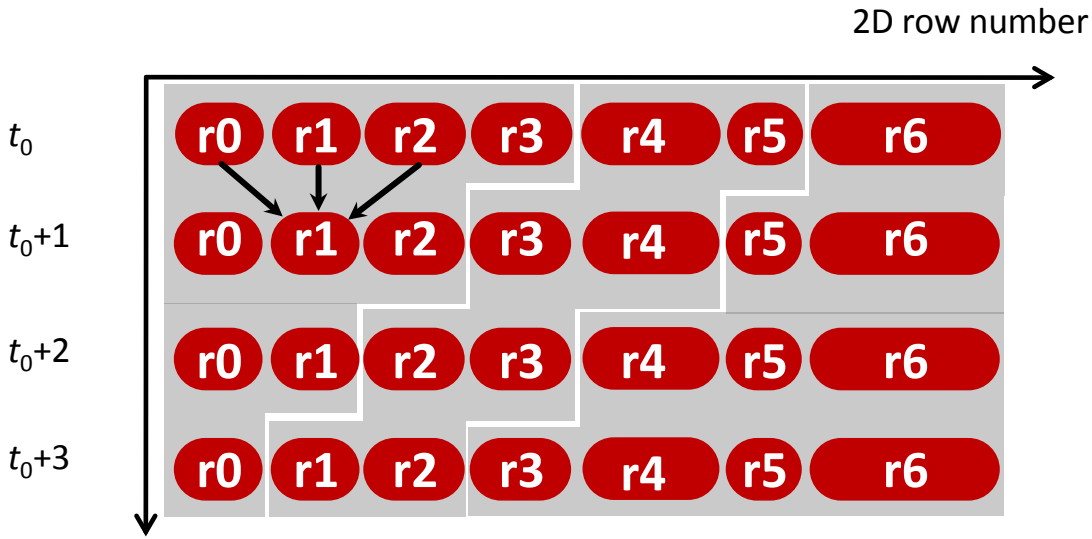
Autotuning parameter: tile size

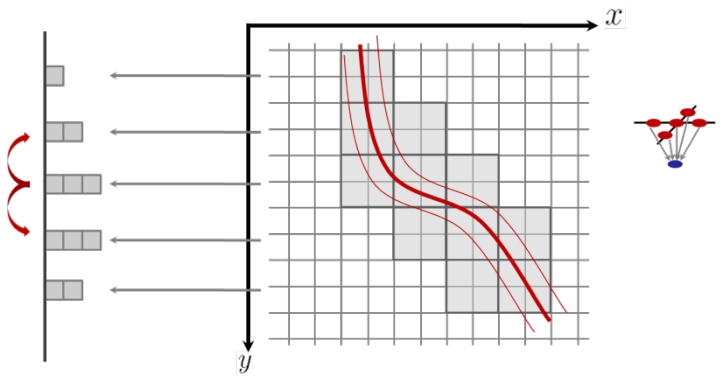
Projective Time Skewing

Standard 1-D dense time-skewing



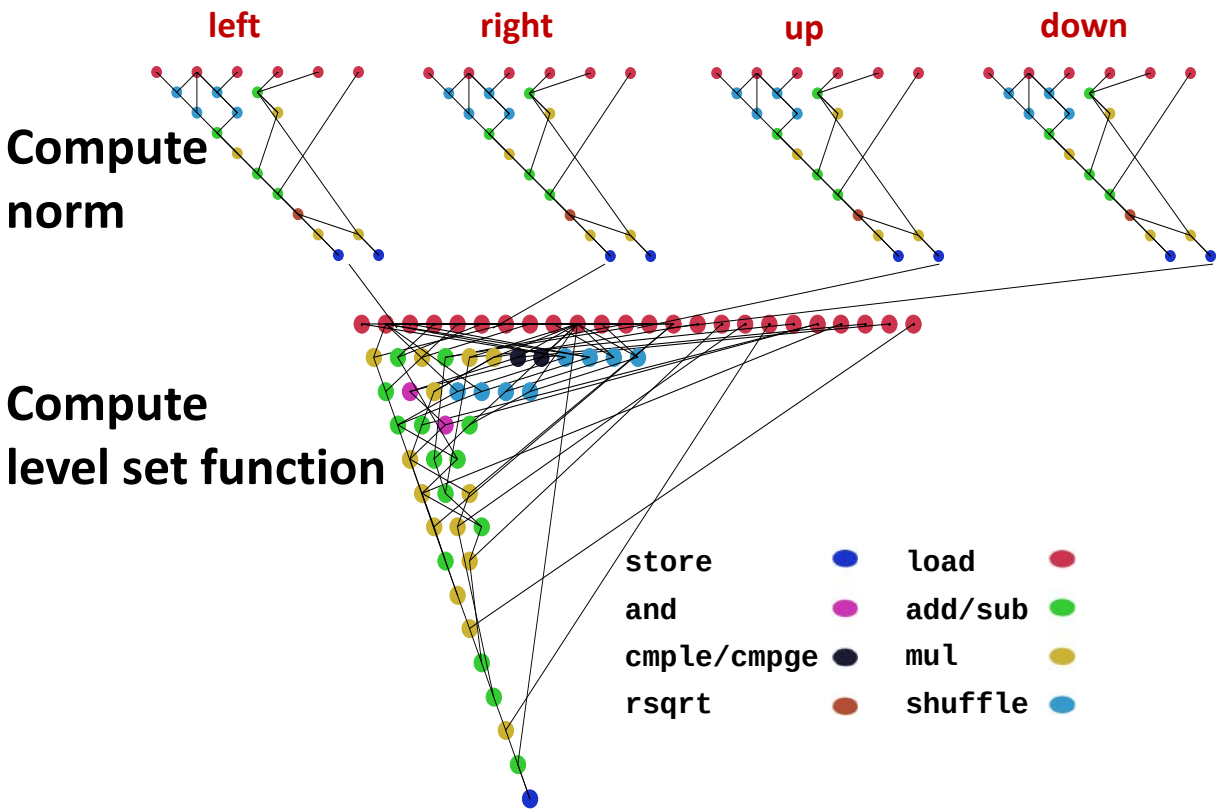
Projective time-skewing



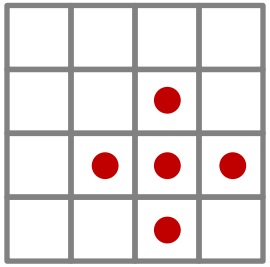


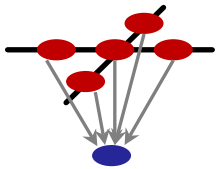
Autotuning parameter: polyhedral tile size

In-Core Stencil Optimization

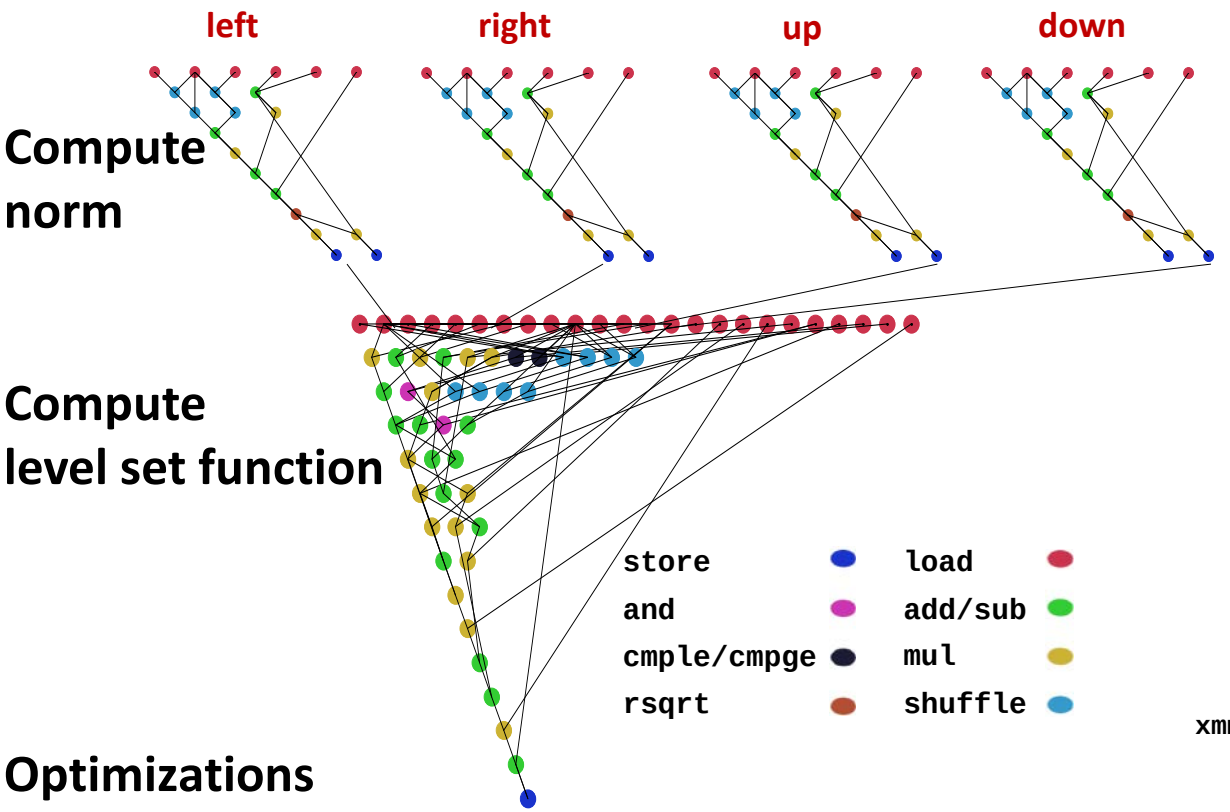


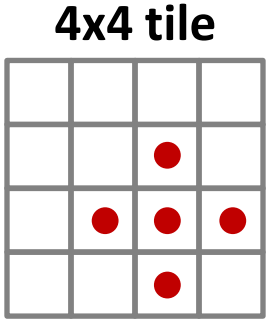
4x4 tile

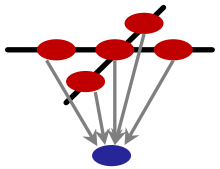


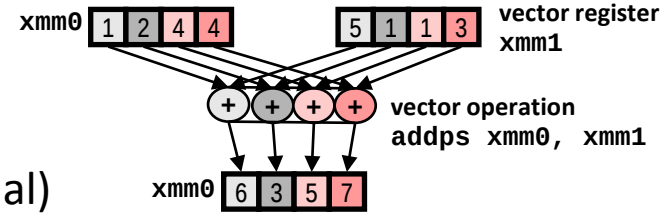


In-Core Stencil Optimization





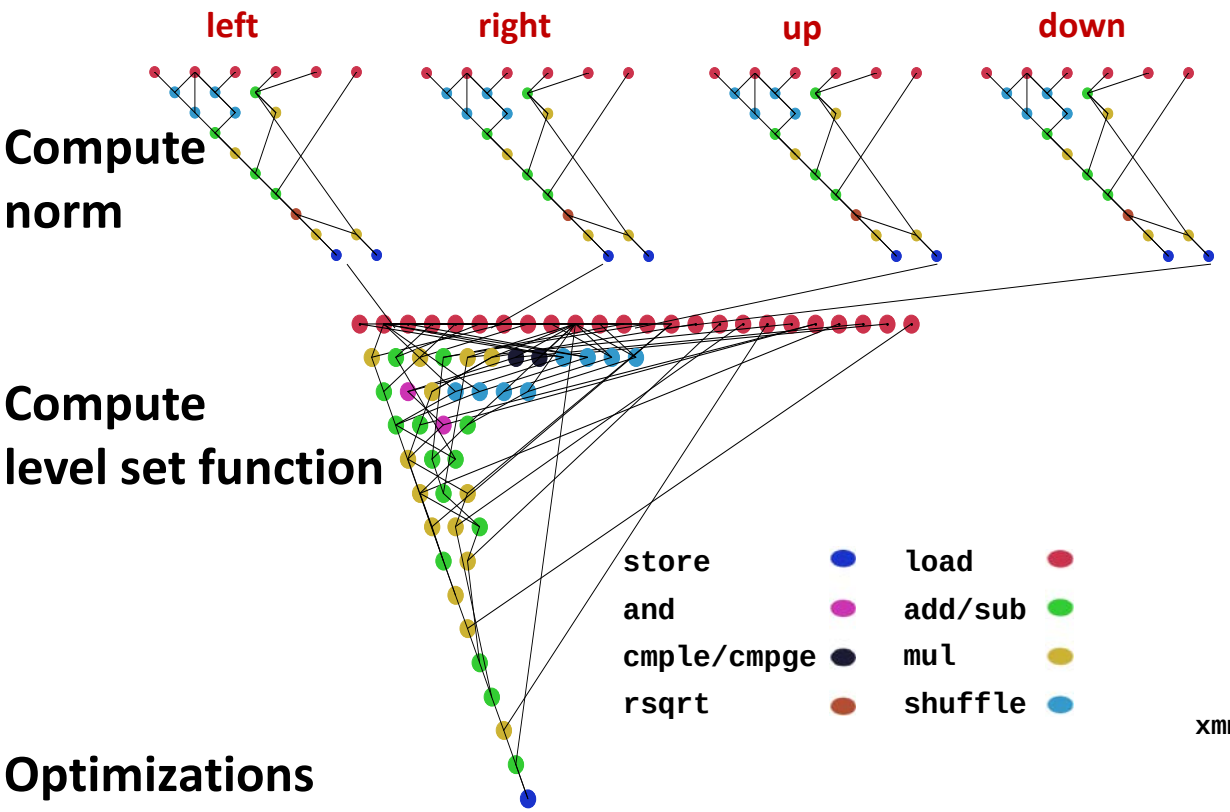


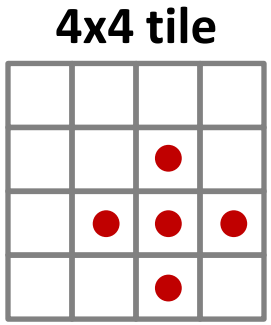


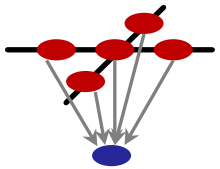
Optimizations

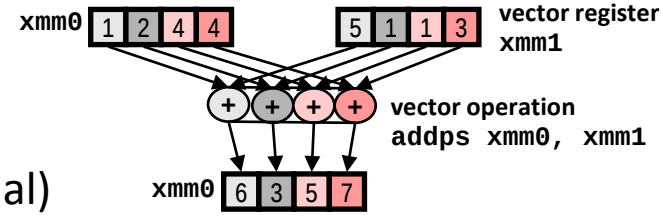
- Common subexpression elimination
- Transcendental function approximation (cos by polynomial)
- SIMD vectorization
- Instruction scheduling (intra and inter stencil)

In-Core Stencil Optimization







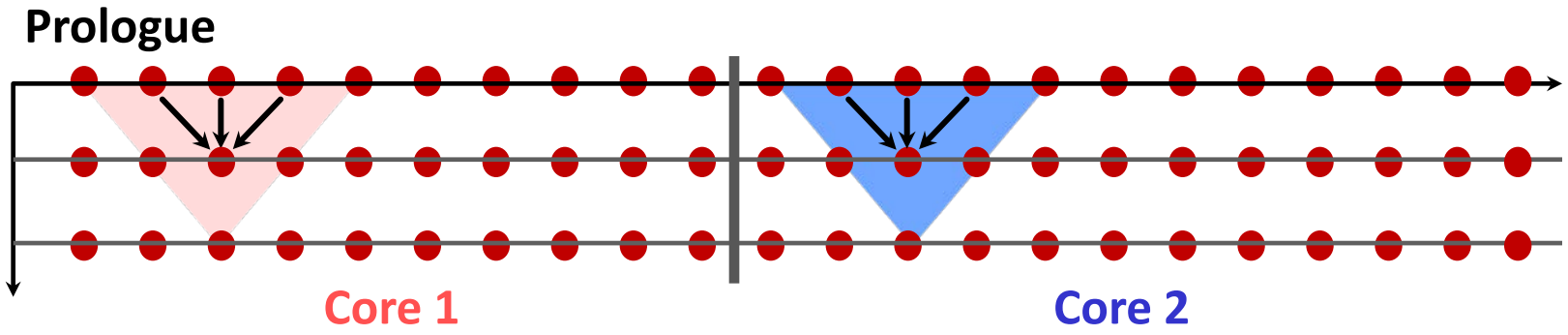


Optimizations

- Common subexpression elimination
- Transcendental function approximation (cos by polynomial)
- SIMD vectorization
- Instruction scheduling (intra and inter stencil)

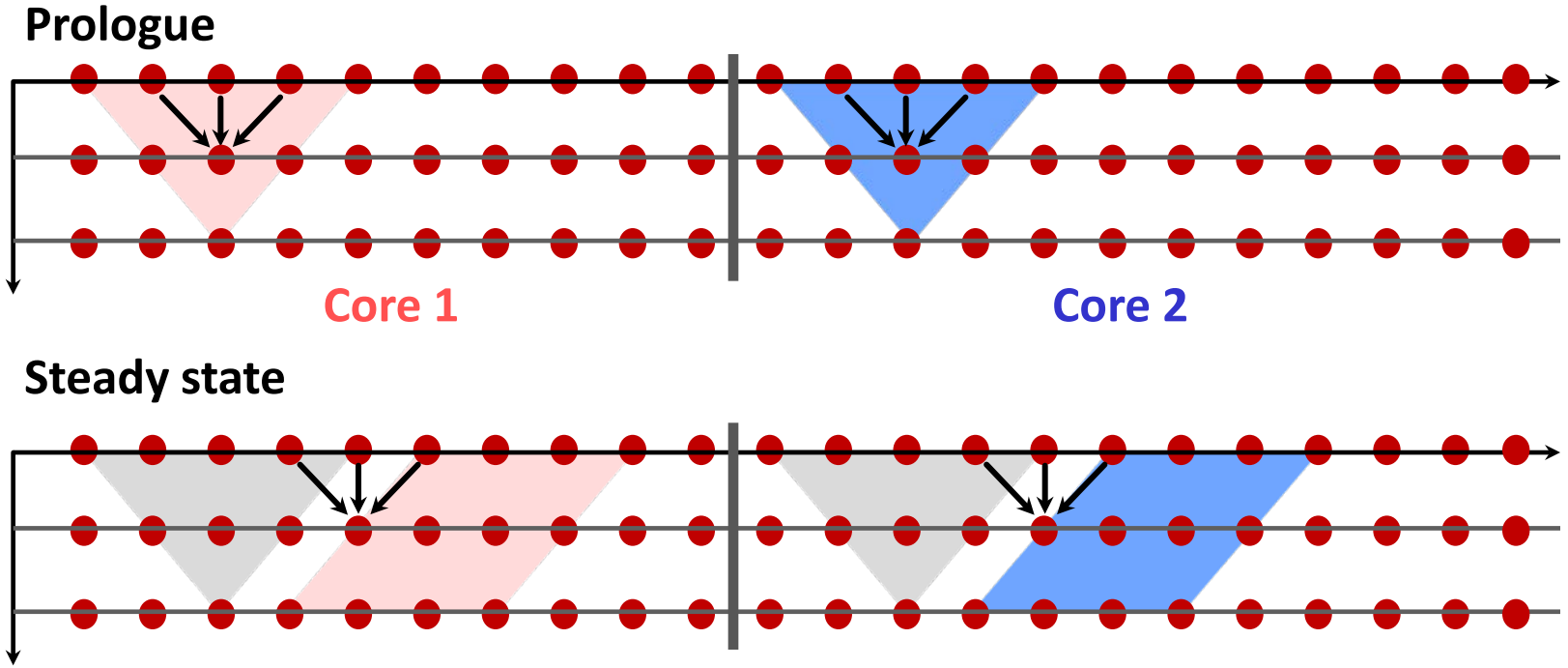
Autotuning parameter: tile instruction schedule

Multicore Optimization: Software Pipeline

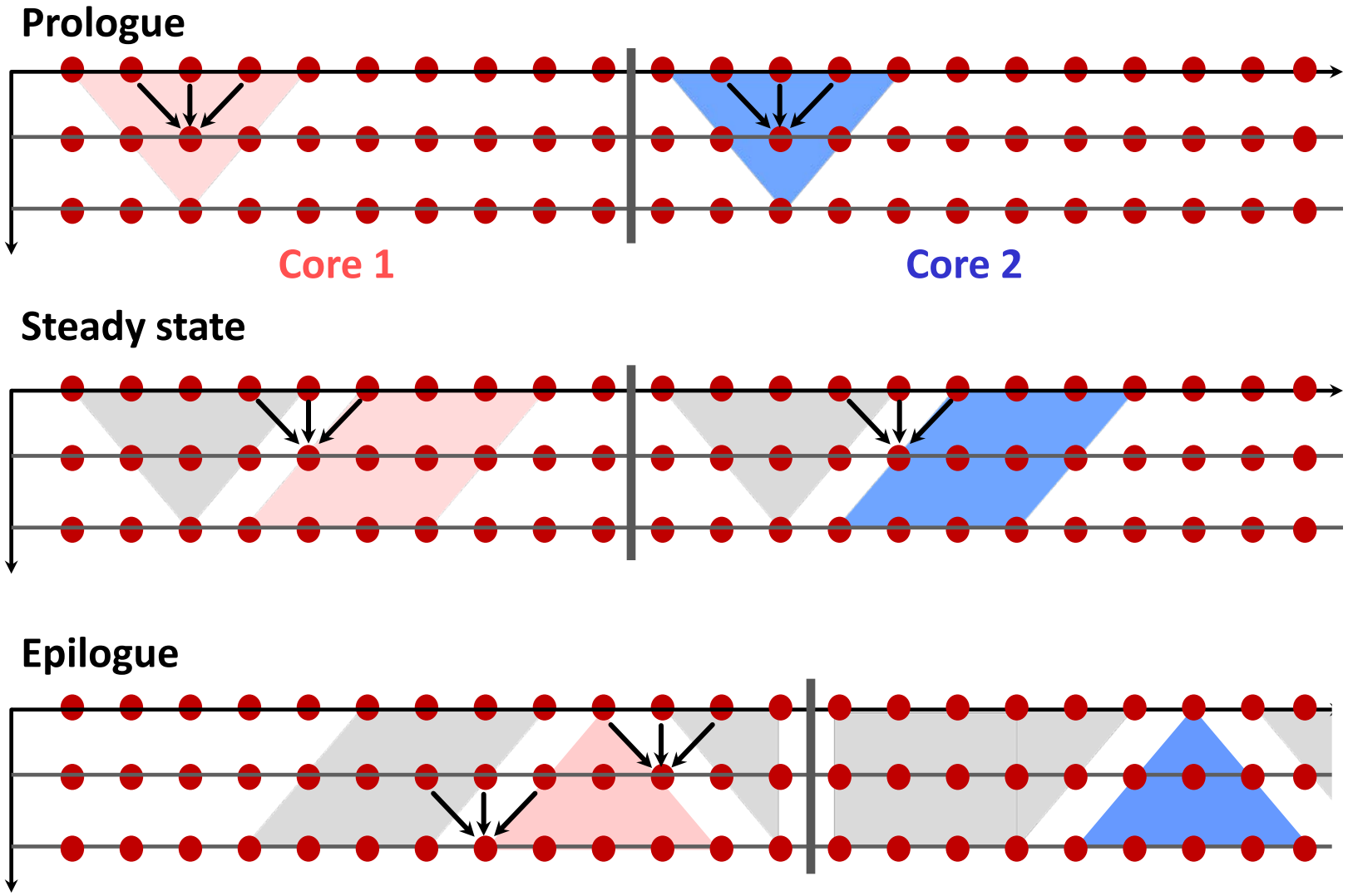


Autotuning parameter: prologue/epilogue vs. steady state

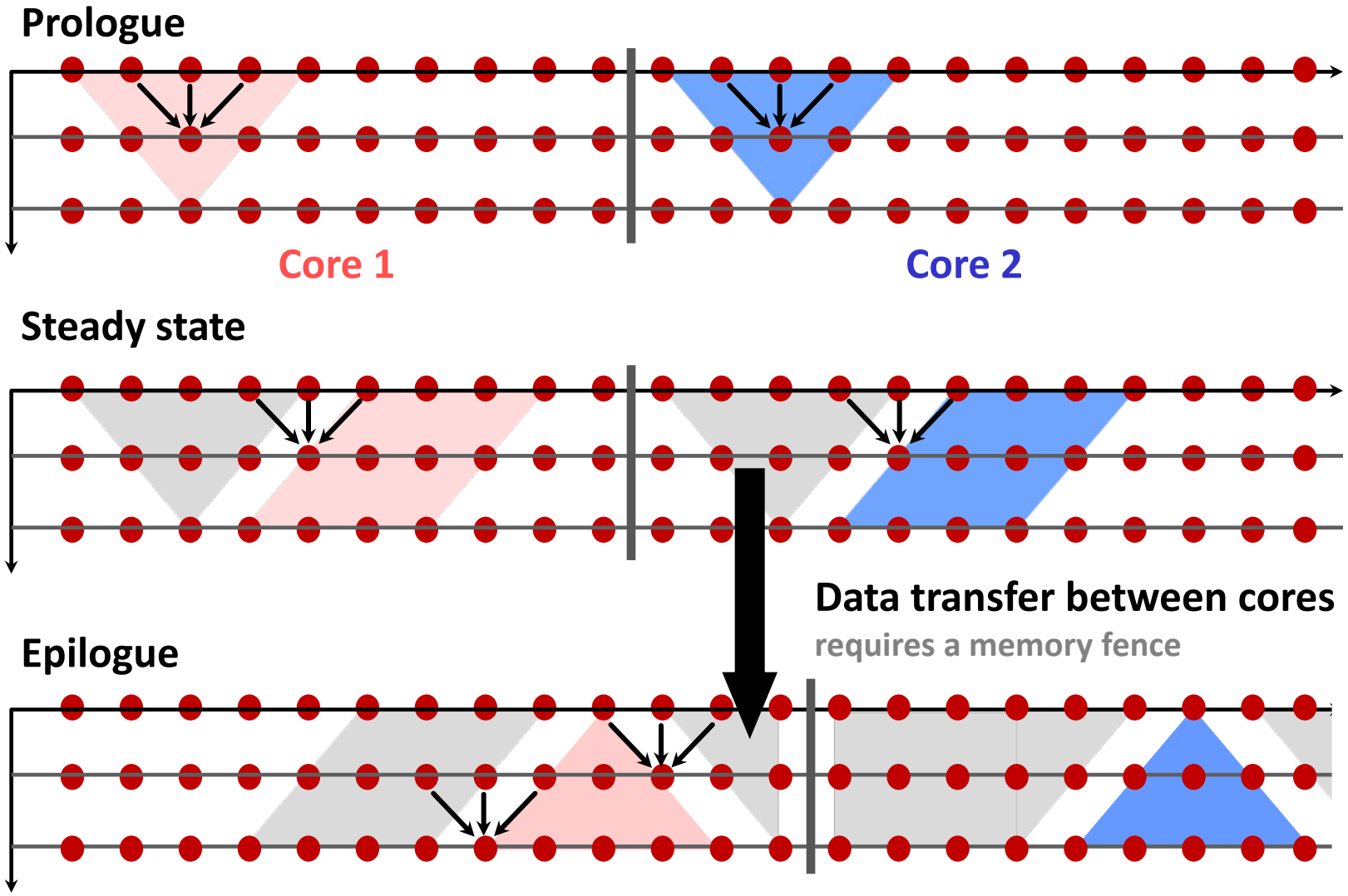
Multicore Optimization: Software Pipeline



Multicore Optimization: Software Pipeline

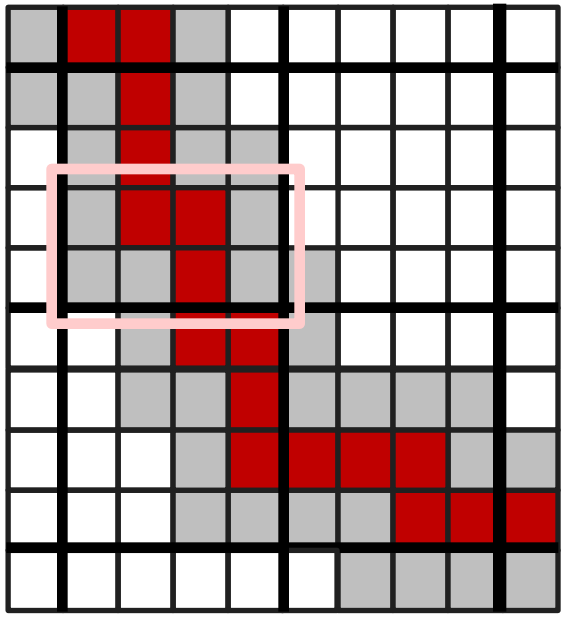


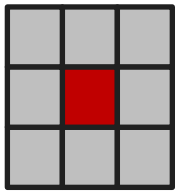
Multicore Optimization: Software Pipeline



Autotuning parameter: prologue/epilogue vs. steady state

Narrow Band Update: Program Generation

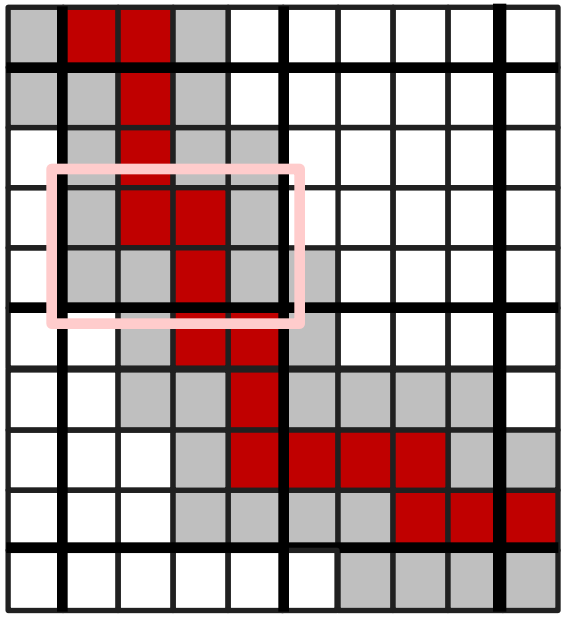


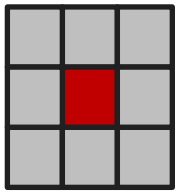


Rubber stamp

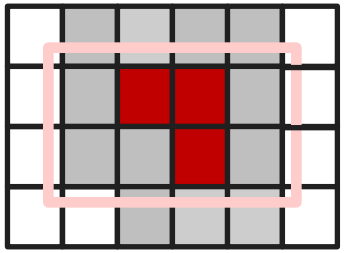
```
// rubber stamp zero level set
for (i=0;i<TILE;i++)
  for (j=0;j<TILE;j++)
    if (!is_zero(Ti+i,Tj+j)) {
      for (k=-BAND;k<=BAND;k++)
         for (m=-BAND;m<=BAND;m++)
           band[Ti+i+k][Tj+j+m]=1;
    }
```

Narrow Band Update: Program Generation





Rubber stamp

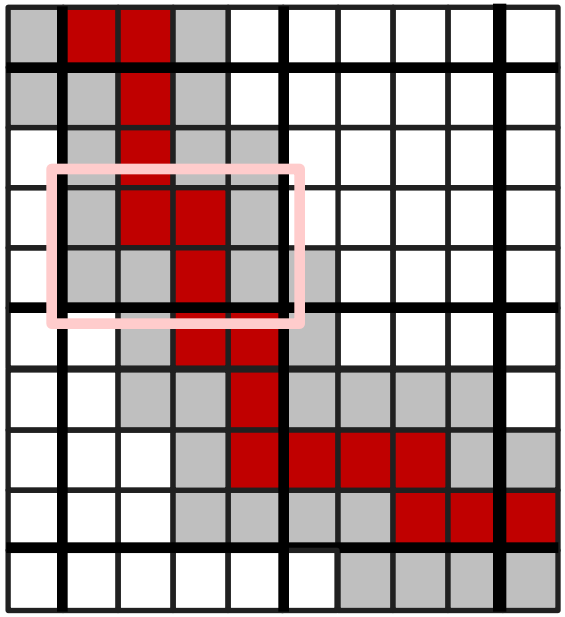


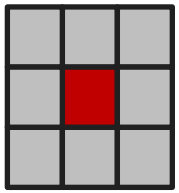
2x4 rubber stamp

```
// rubber stamp zero level set
for (i=0;i<TILE;i++)
  for (j=0;j<TILE;j++)
    if (!is_zero(Ti+i,Tj+j)) {
      for (k=-BAND;k<=BAND;k++)
         for (m=-BAND;m<=BAND;m++)
           band[Ti+i+k][Tj+j+m]=1;
    }
```

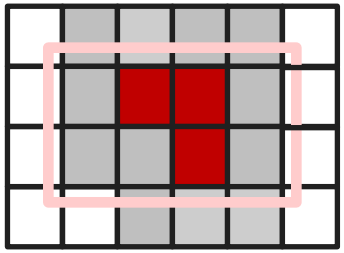
```
// TILE=4x4, BAND=+/-1
// unroll: 2x4
for (i=0;i<4;i+=2) {
  b0=&band[Ti+i-1][Tj-1];
  b1=&band[Ti+i][Tj-1];
  b2=&band[Ti+i+1][Tj-1];
  b3=&band[Ti+i+2][Tj-1];
  switch (zeropat(Ti+i,Tj)) {
    ...
    case PAT(0,1,1,0
             0,0,1,0):
      b0[1]=1;b0[2]=1;b0[3]=1;b0[4]=1;
      b1[1]=1;b1[2]=1;b1[3]=1;b1[4]=1;
      b2[1]=1;b2[2]=1;b2[3]=1;b2[4]=1;
      b3[2]=1;b3[3]=1;b2[4]=1;
      break;
    case PAT(...):
}
```

Narrow Band Update: Program Generation





Rubber stamp



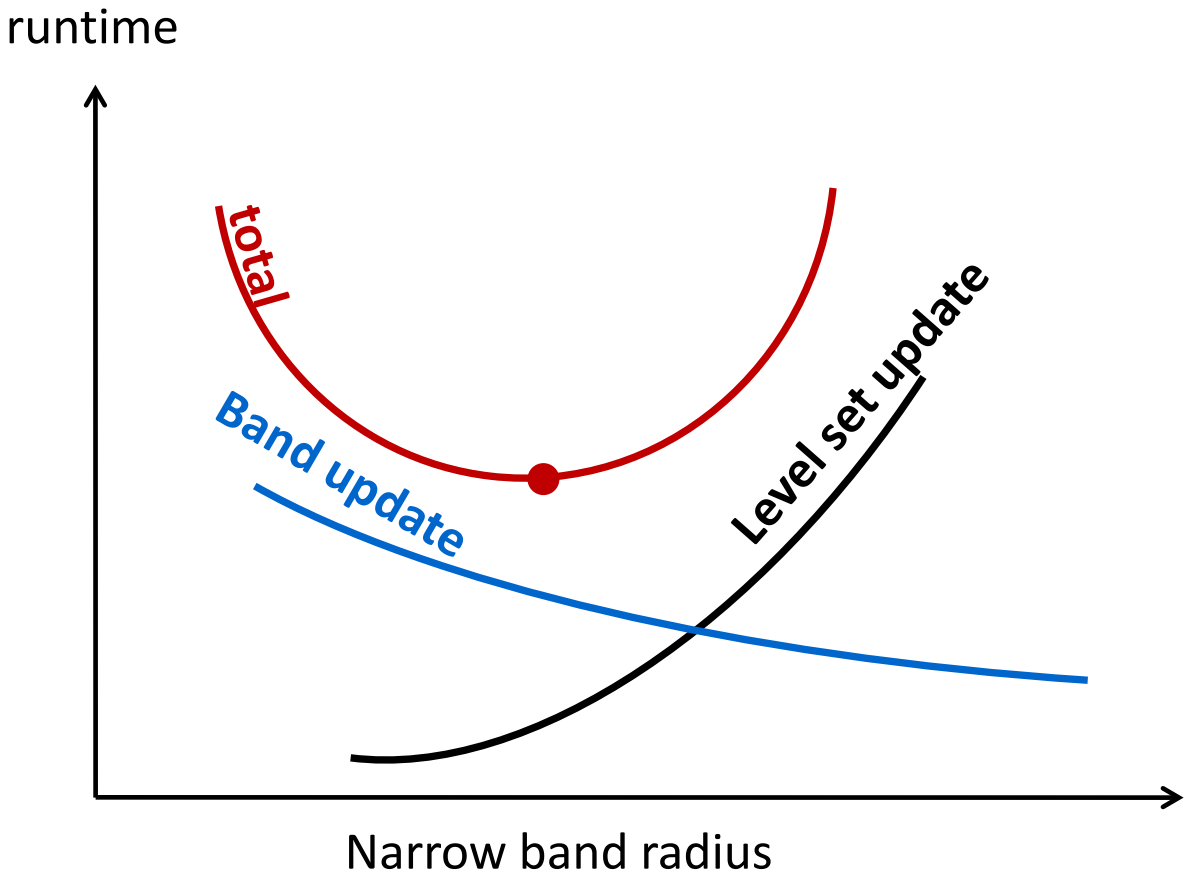
2x4 rubber stamp

```
// rubber stamp zero level set
for (i=0;i<TILE;i++)
  for (j=0;j<TILE;j++)
    if (!is_zero(Ti+i,Tj+j)) {
      for (k=-BAND;k<=BAND;k++)
         for (m=-BAND;m<=BAND;m++)
           band[Ti+i+k][Tj+j+m]=1;
    }
```

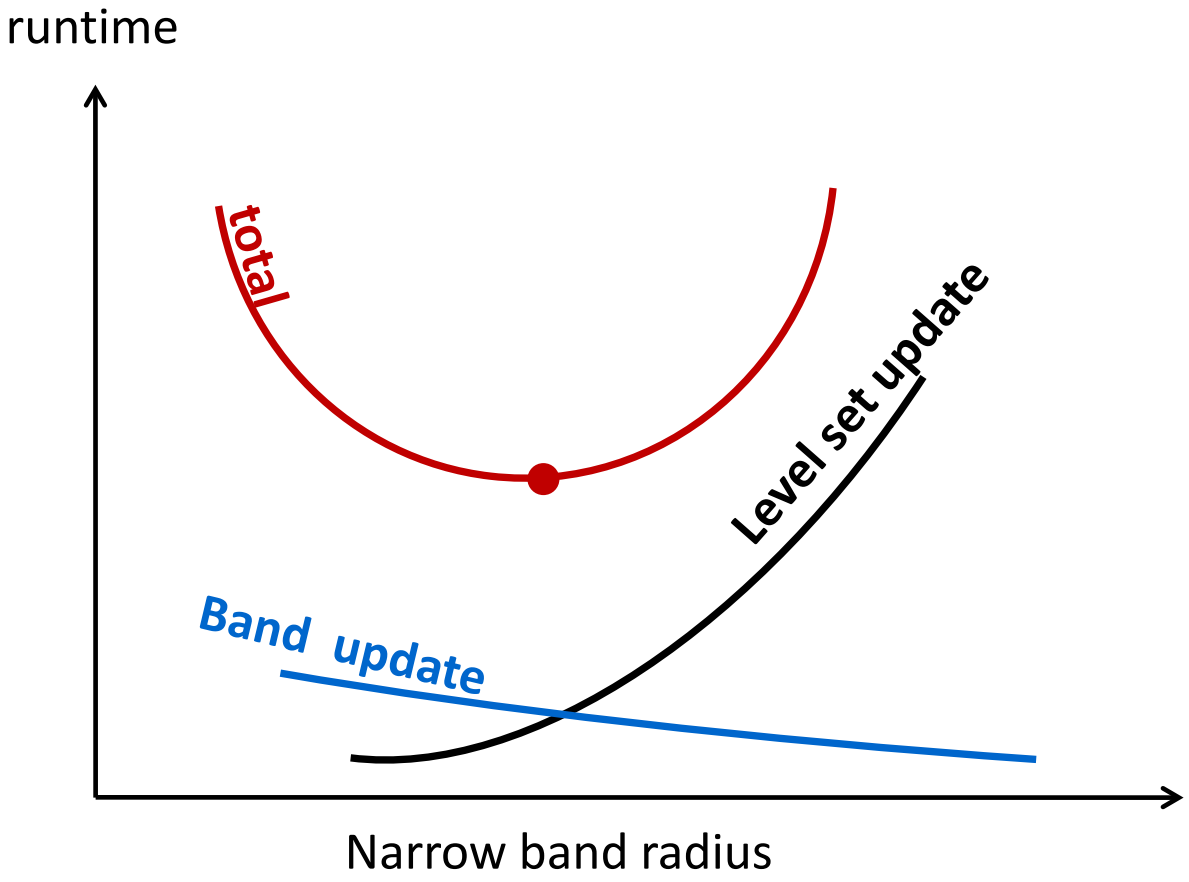
```
// TILE=4x4, BAND=+/-1
// unroll: 2x4
for (i=0;i<4;i+=2) {
  b0=&band[Ti+i-1][Tj-1];
  b1=&band[Ti+i][Tj-1];
  b2=&band[Ti+i+1][Tj-1];
  b3=&band[Ti+i+2][Tj-1];
  switch (zeropat(Ti+i,Tj)) {
    ...
    case PAT(0,1,1,0
             0,0,1,0):
      b0[1]=1;b0[2]=1;b0[3]=1;b0[4]=1;
      b1[1]=1;b1[2]=1;b1[3]=1;b1[4]=1;
      b2[1]=1;b2[2]=1;b2[3]=1;b2[4]=1;
      b3[2]=1;b3[3]=1;b2[4]=1;
      break;
    case PAT(...):
}
```

Autotuning parameter: rubber stamp unrolling

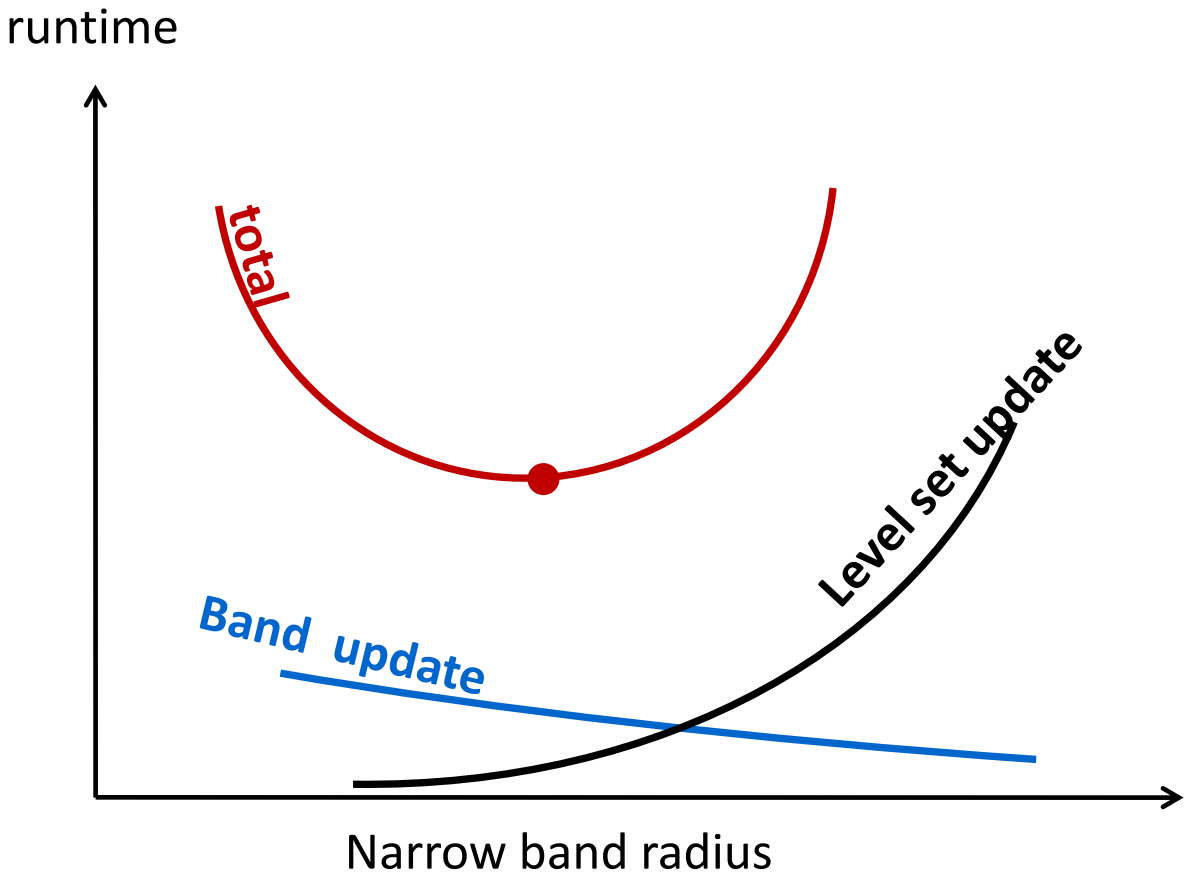
A Fundamental Tradeoff



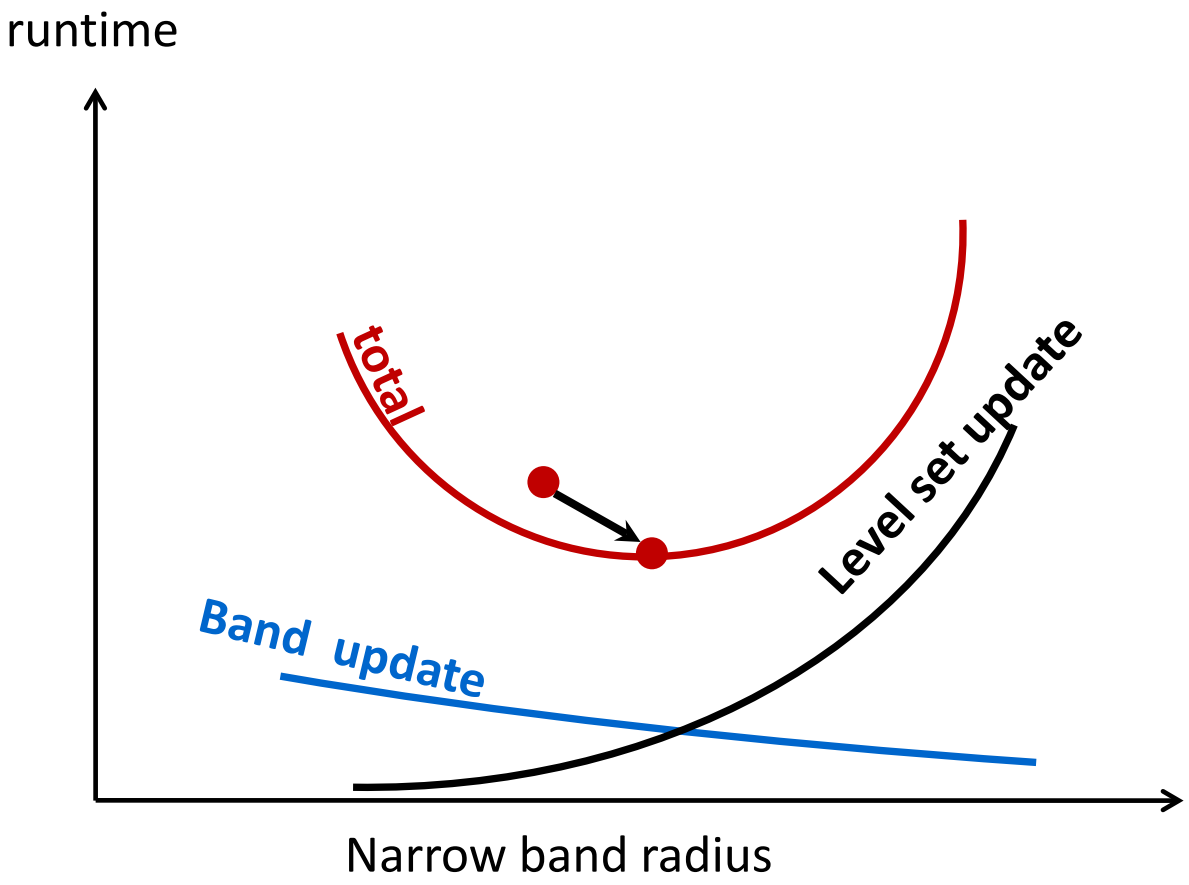
A Fundamental Tradeoff



A Fundamental Tradeoff



A Fundamental Tradeoff



Parameter Space

Tuning Parameters	Range
Tile size	{1,2,4,8}x{4,8,16}
Band radius	{1,2,3,4,5,6,7,8}
Polytope size height	{2, 4, 8, 16, 24, 32, 48, 64, 80, 96, 128, 160, 192}
Polytope size height	{2, 4, 8, 12, 16, 32}
Enable/disable	SIMD, instruction scheduling, approx. cos(), padding, large pages,…

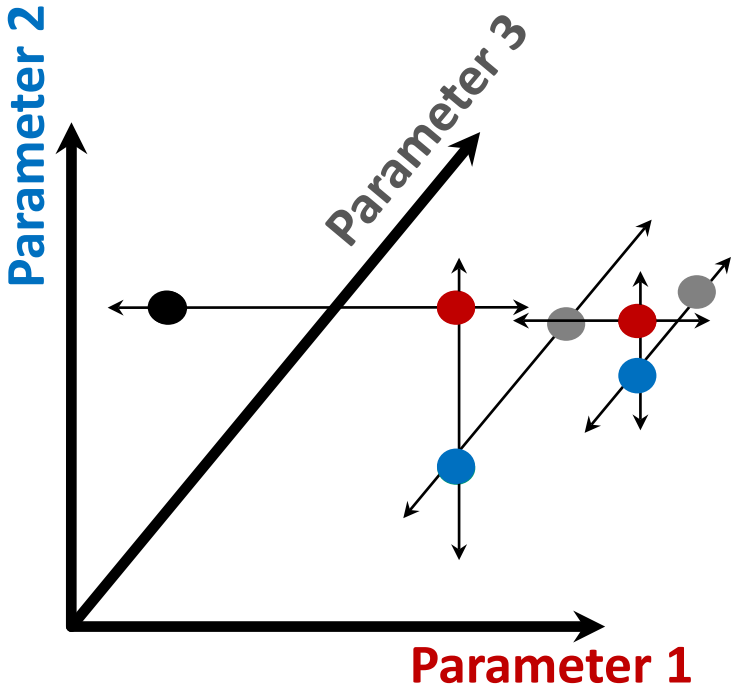
Autotuning: Iterative Line Search

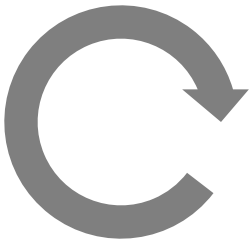
Iterative Line search

- Search parameters one by one hold others fixed
- Multiple refinement iterations
- Search space is well-behaved

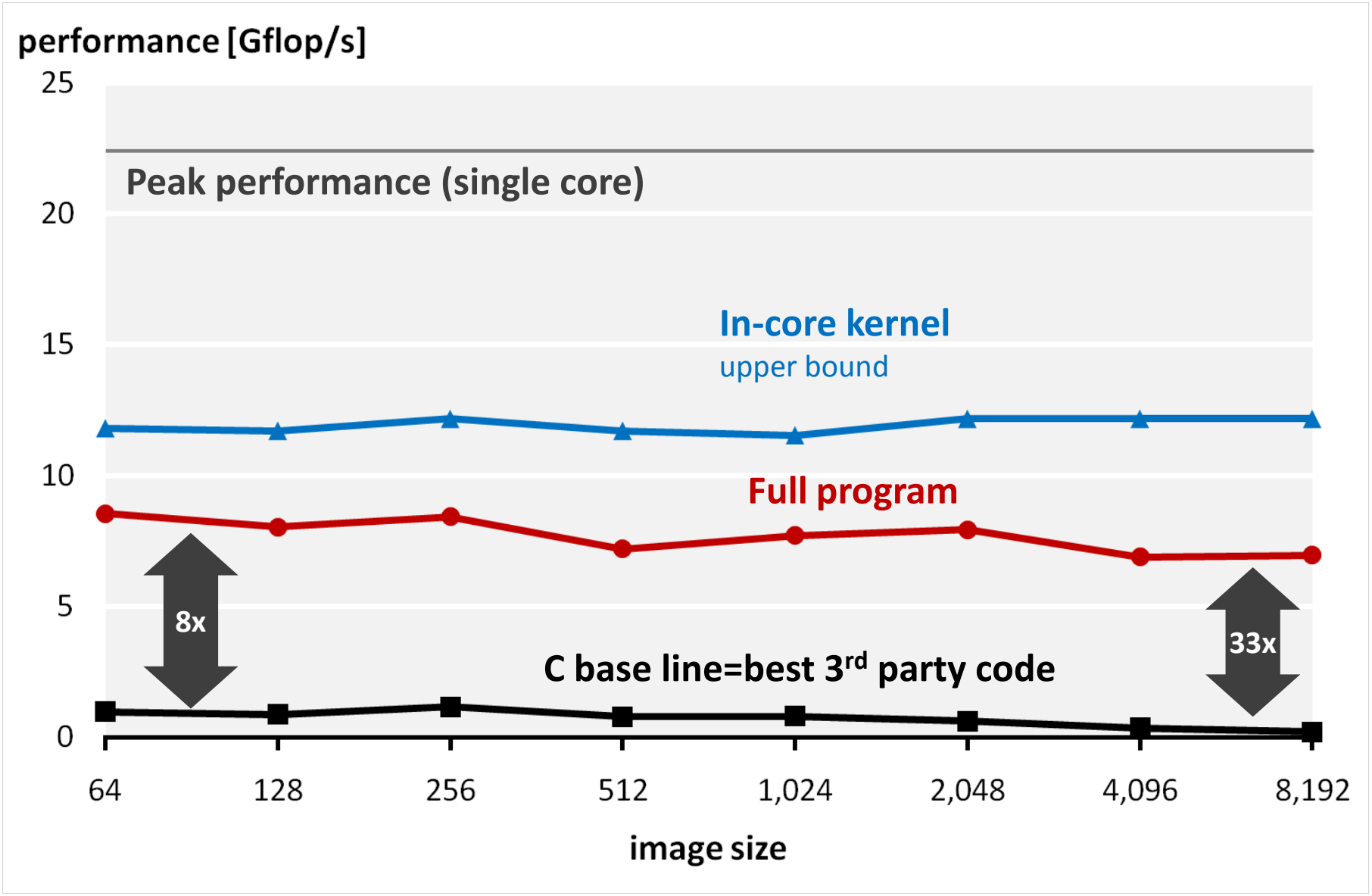
Heuristics

- Cut search space
- For instance: DAG scheduling

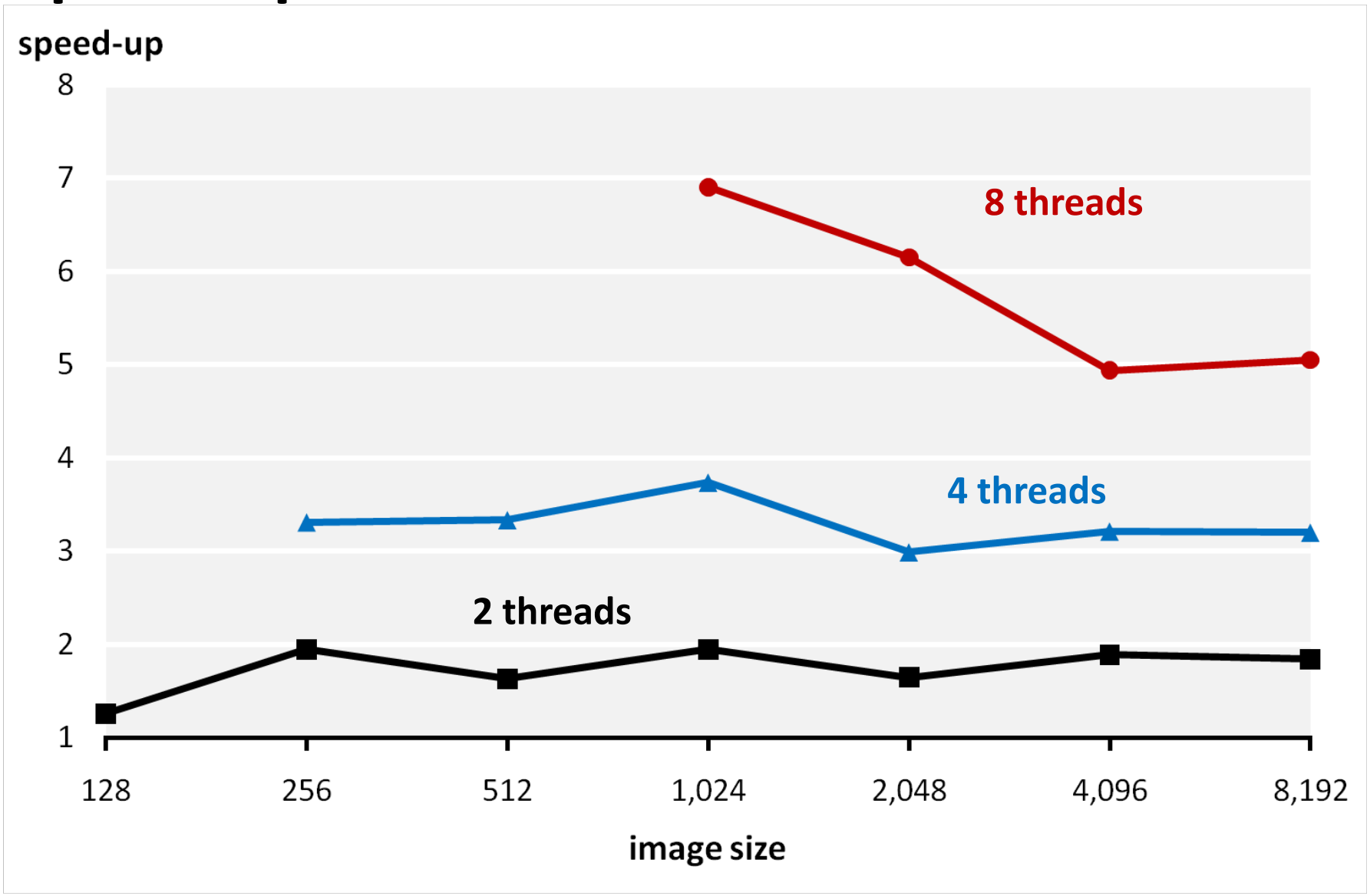




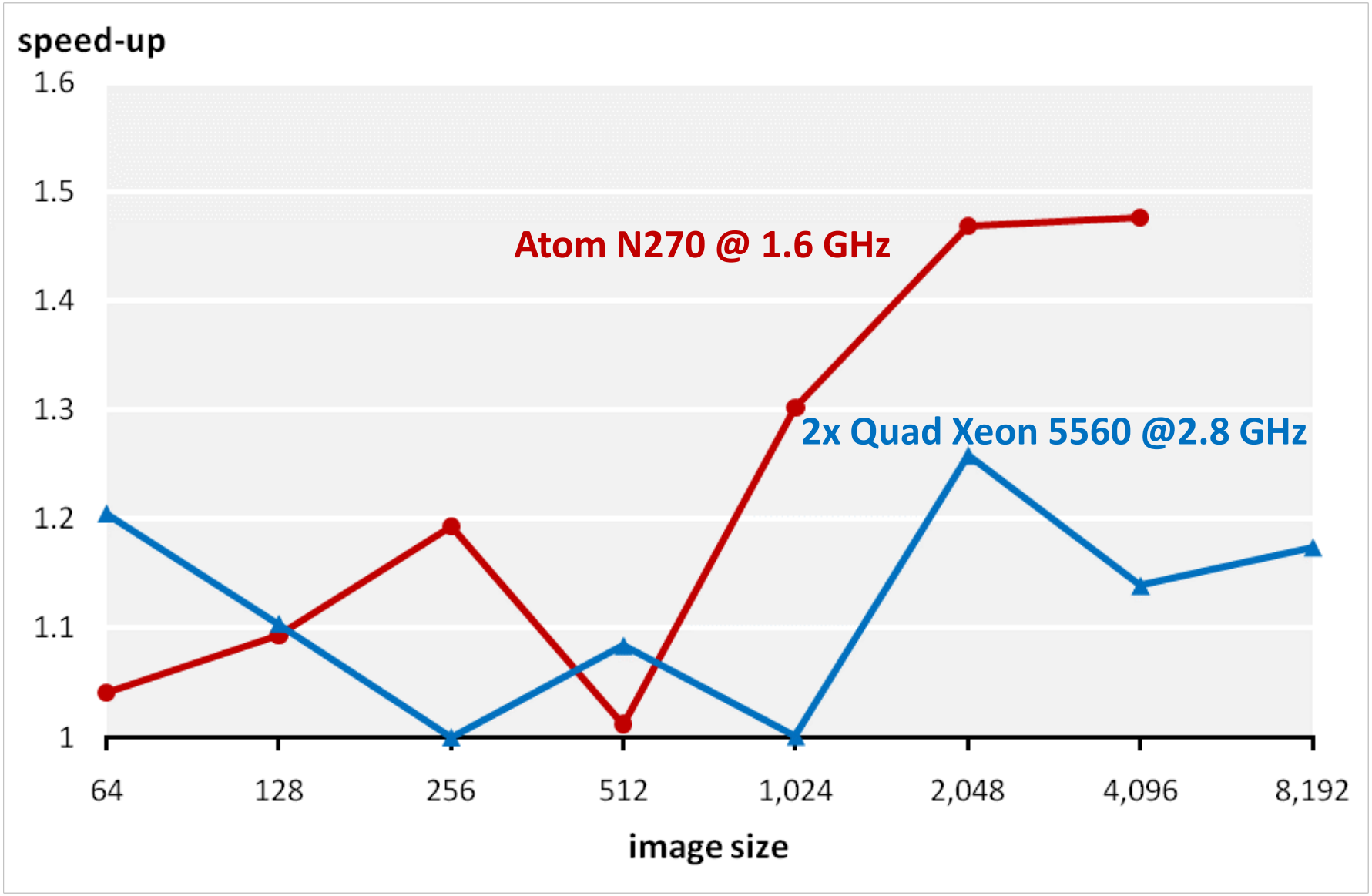
Efficiency on Single Core 2.8 GHz Xeon 5560



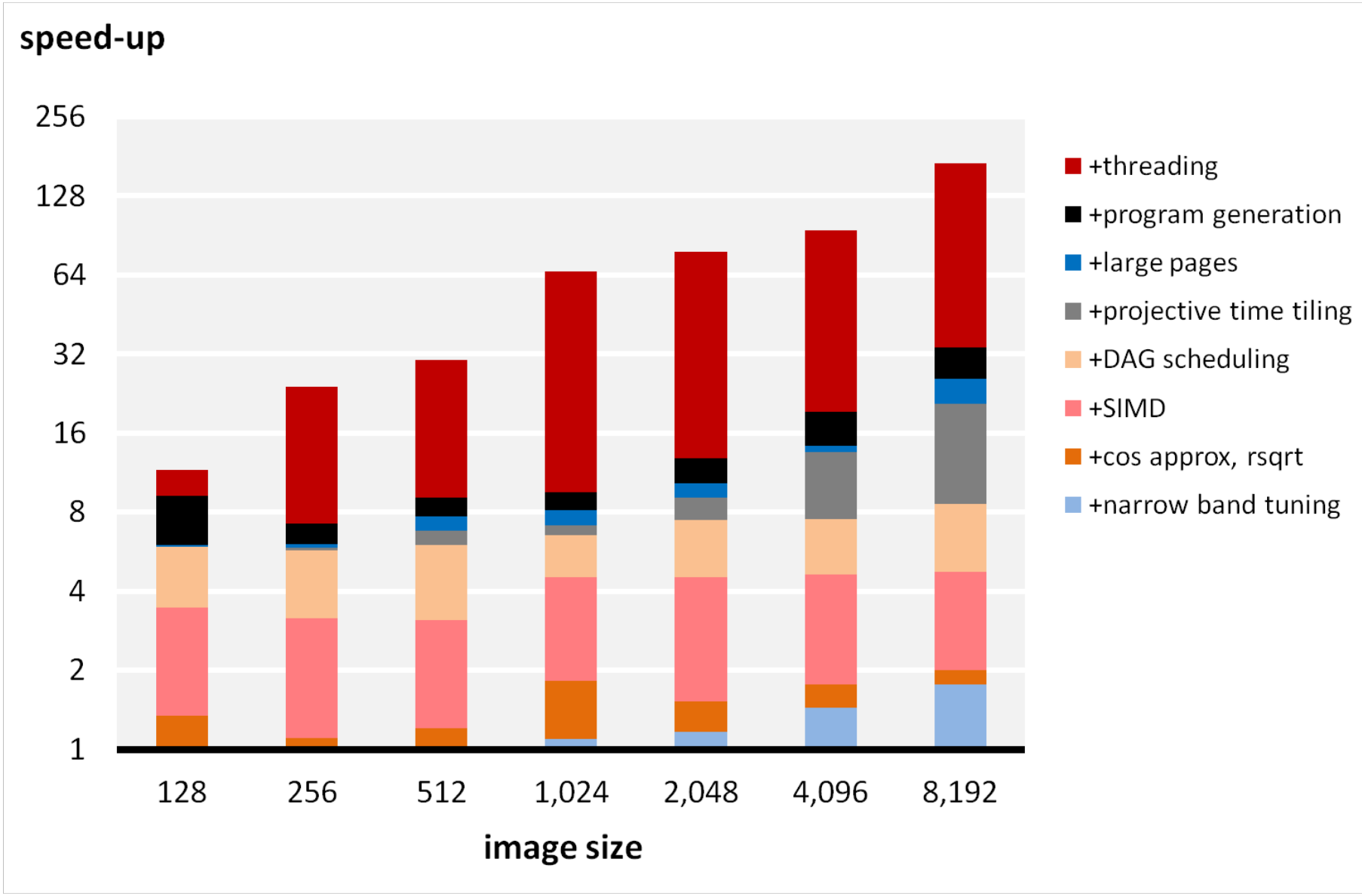
Speed-Up On 2x 2.8 GHz Quad Xeon 5560



Performance Gain Through Autotuning



Speedup Breakdown on 2x Xeon 5560



Lessons From Optimizing Level Set

- **10x to 100x speed-up is often possible**
  really hard work, specific to the problem
- **Doable with standard tools**
  C compiler, autotuning, and program generation scripts
- **Crucial: extract meaningful parameters**
  capture machine/algorithm interaction, one algorithm a time
- **Many things must be done right**
  SIMD, threading, low-level tricks, specialization, tuning
- **Autotuning gives the last 50% to 2x speed-up**
  the remaining 5x to 50x come from standard optimization

More Information:

www.spiral.net

www.spiralgen.com