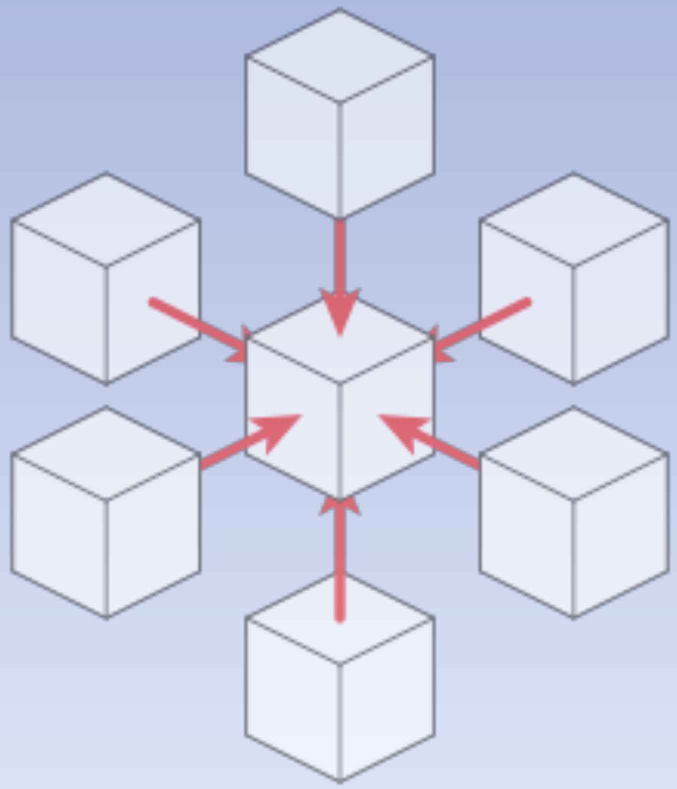
Auto-Generation and Auto-Tuning of 3D Stencil Codes on GPU Clusters



- Yongpeng Zhang, Frank Mueller
- North Carolina State University

- CGO‘2012

NC STATE UNIVERSITY
Department of Computer Science

Outline

- Motivation
- DSL front-end and Benchmarks
- Framework
- Experimental Results
- Conclusion

Motivation

- Main-stream microprocessors more parallel
  - GPU becomes a popular accelerator nowadays
- Achieving near-optimal perf. on GPUs still difficult
  - Perf. affected by a multitude of architectural features -- tradeoffs
  - Architectural difference b/w generations of hw line
  - Hand-tuning for all optimization options? -- counter-productive
- Domain Specific Languages (HTML, MATLAB, SQL)
  - Balance portability, performance and programmability
  - Sacrifice language generality
  - Performance comparable to hand-coded code

DSL for 3D Stencil

- Portable source-to-source (DSL to CUDA) framework
  - Auto-generation and auto-tuning for different GPUs
  - Iterative 3D Jacobi stencil computation
- Based on prior research to accelerate 3D stencil on GPUs
  - Summarize optimization techniques
- Abstract stencil into domain-specific specifications
- Extract critical tuning parameters
- Close to hand-written code

Related Work

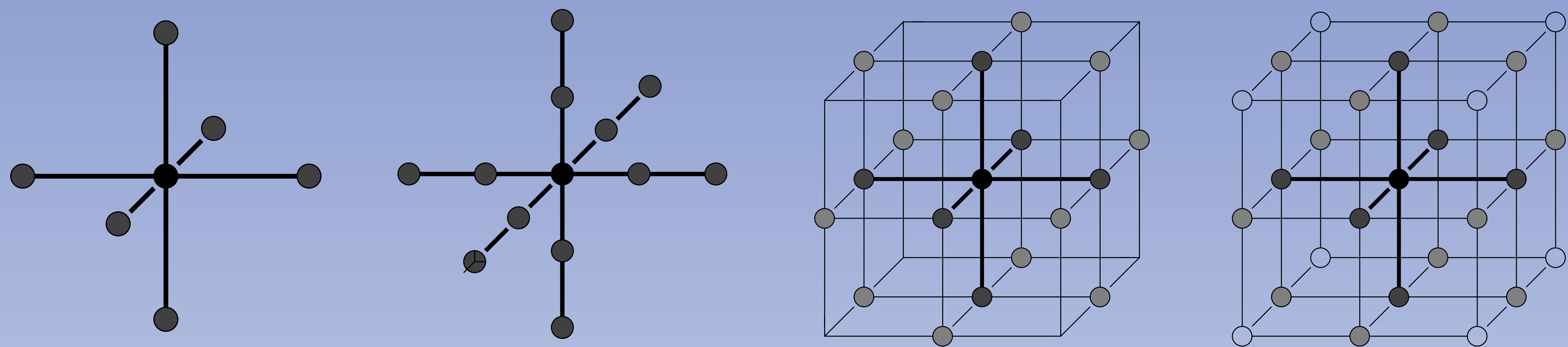
- Auto-tuning: portability and productivity
  - Libraries: ATLAS [2], OSKI [23], FFTW [6]
- Auto-tuning on GPUs
  - Sparse Matrix-vector multiplication [7]
  - GEMM [12], 3D FFT [19]
- Stencil on GPUs
  - Hand-coded ([17, 18, 20])
  - Ease of programming (auto-generation) ([5, 11, 14, 22])
  - Tuning one parameter ([13, 16])
- Our work offers both performance and programmability
  - Code auto-generation and auto-tuning

Stencil DSL

$$\begin{aligned} out([i][j][k]) &= \sum_m w_m * in[i \pm I_m][j \pm J_m][k \pm K_m] \\ &+ \sum_l w_l[i][j][k] * in[i \pm I_l][j \pm J_l][k \pm K_l] \\ &+ \sum_n w_n * in_n \end{aligned} \tag{1}$$

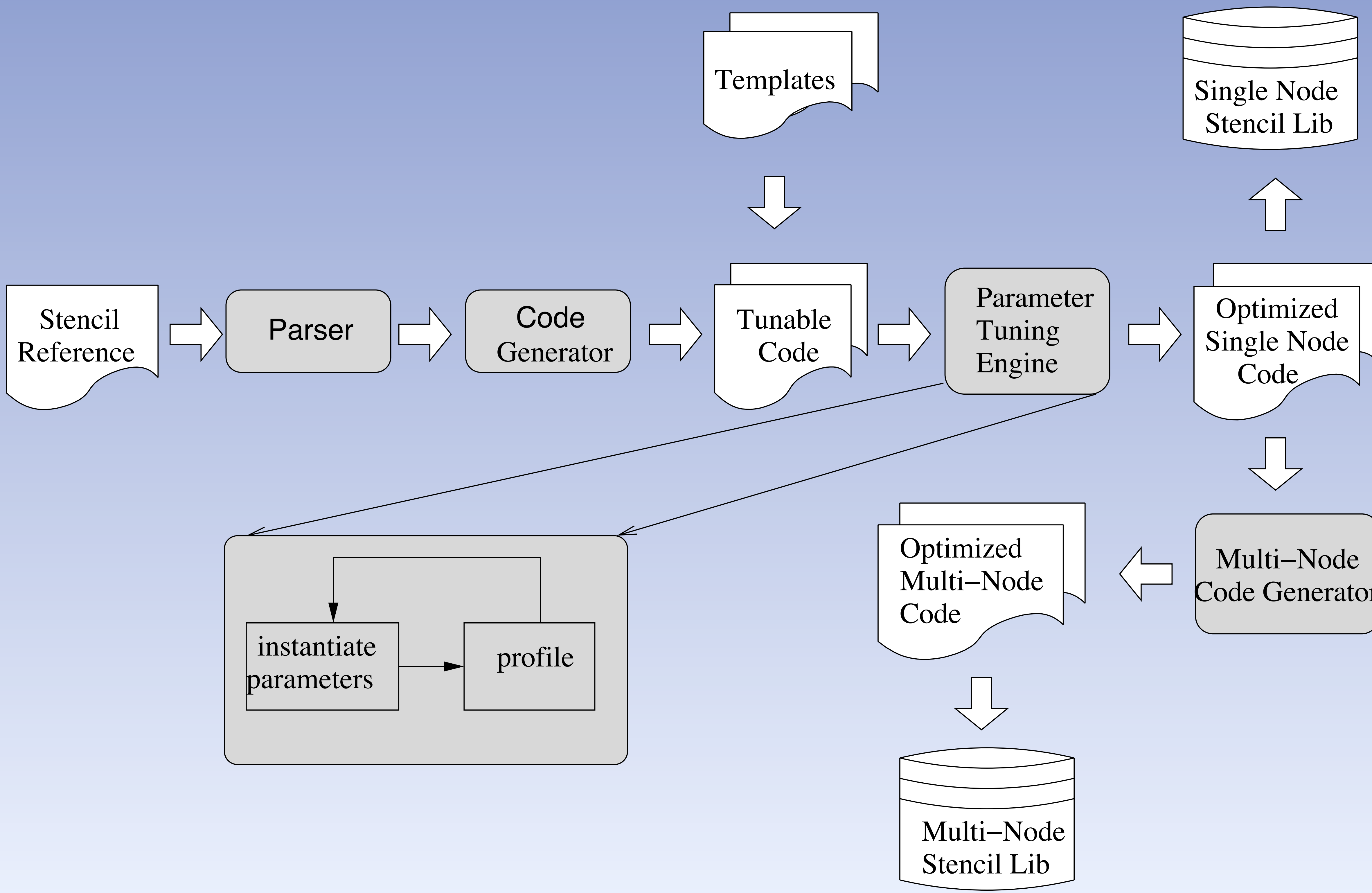
- A sequence of such equations
- Single output array *(out[i][j][k])*
- Single input array *(in[i + I][j + J][k + K])*
- Multiple input parameter array *(w[i][j][k])*
- *Multiple temporary variable (out)*

Benchmarks (7/13/19/27 point)

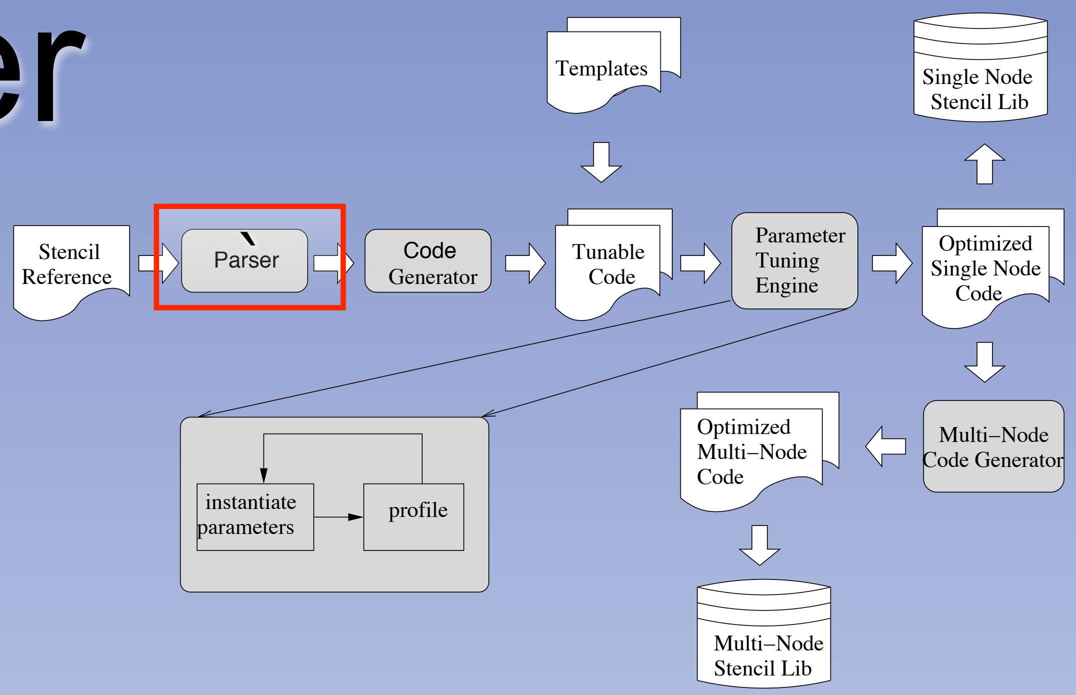


Kernel	Specification
7-point order-1	$tmp = (u_{i+1,j,k} + u_{i-1,j,k} + u_{i,j+1,k} + u_{i,j-1,k} + u_{i,j,k+1} + u_{i,j,k-1}) * beta;$ $u1_{i,j,k} = tmp + alpha * u_{i,j,k};$
13-point order-2	$tmp = coef1 * (u_{i+1,j,k} + u_{i-1,j,k} + u_{i,j+1,k} + u_{i,j-1,k} + u_{i,j,k+1} + u_{i,j,k-1});$ $tmp+ = coef2 * (u_{i+2,j,k} + u_{i-2,j,k} + u_{i,j+2,k} + u_{i,j-2,k} + u_{i,j,k+2} + u_{i,j,k-2});$ $u1_{i,j,k} = tmp + coef0 * u_{i,j,k};$
19-point order-1 (himeno)	$s0 = wrk1_{i,j,k} + a0d_{i,j,k} * p_{i,j,k+1} + a1d_{i,j,k} * p_{i,j+1,k+1};$ $s0+ = b0d_{i,j,k} * (p_{i,j+1,k+1} - p_{i,j-1,k+1} - p_{i,j+1,k-1} + p_{i,j-1,k-1}) + a2d_{i,j,k} * p_{i+1,j,k};$ $s0+ = b1d_{i,j,k} * (p_{i+1,j+1,k} - p_{i-1,j+1,k} - p_{i+1,j-1,k} + p_{i-1,j-1,k});$ $s0+ = b2d_{i,j,k} * (p_{i+1,j,k+1} - p_{i-1,j,k+1} - p_{i+1,j,k-1} + p_{i-1,j,k-1}) + c0d_{i,j,k} * p_{i,j,k-1};$ $s0+ = c1d_{i,j,k} * p_{i,j-1,k} + c2d_{i,j,k} * p_{i-1,j,k};$ $ss = (s0 * a3d_{i,j,k} - p_{i,j,k}) * bnd_{i,j,k};$ $wrk2_{i,j,k} = p_{i,j,k} + omega * ss;$
27-point order-1	$b_{i,j,k} = param0 * a_{i,j,k}$ $+param1 * (a_{i-1,j,k} + a_{i+1,j,k} + a_{i,j-1,k} + a_{i,j+1,k} + a_{i,j,k-1} + a_{i,j,k+1})$ $+param2 * (a_{i-1,j-1,k} + a_{i-1,j+1,k} + a_{i+1,j-1,k} + a_{i+1,j+1,k} + a_{i-1,j,k-1} + a_{i-1,j,k+1}$ $+a_{i+1,j,k-1} + a_{i+1,j,k+1} + a_{i,j-1,k-1} + a_{i,j-1,k+1} + a_{i,j+1,k-1} + a_{i,j+1,k+1})$ $+param3 * (a_{i-1,j-1,k-1} + a_{i-1,j-1,k+1} + a_{i-1,j+1,k-1} + a_{i-1,j+1,k+1}$ $+a_{i+1,j-1,k-1} + a_{i+1,j-1,k+1} + a_{i+1,j+1,k-1} + a_{i+1,j+1,k+1});$

System Overview

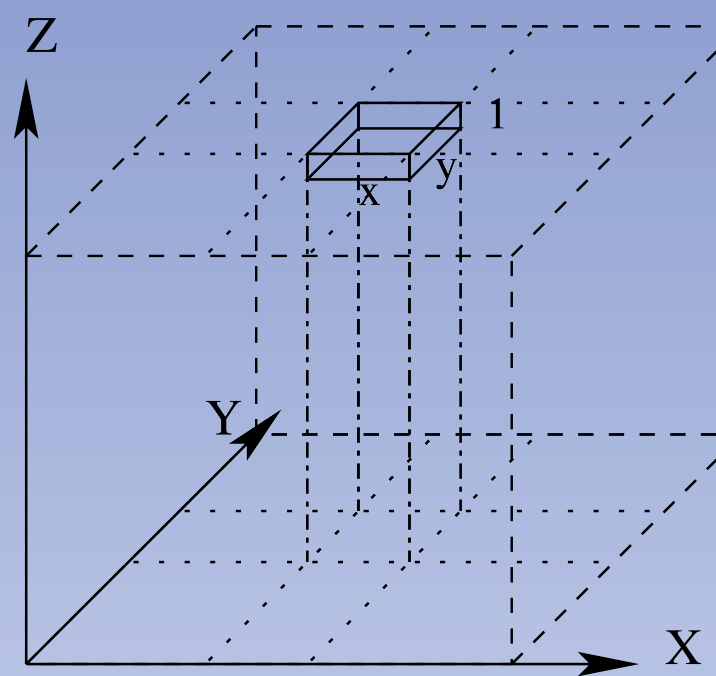


Parser



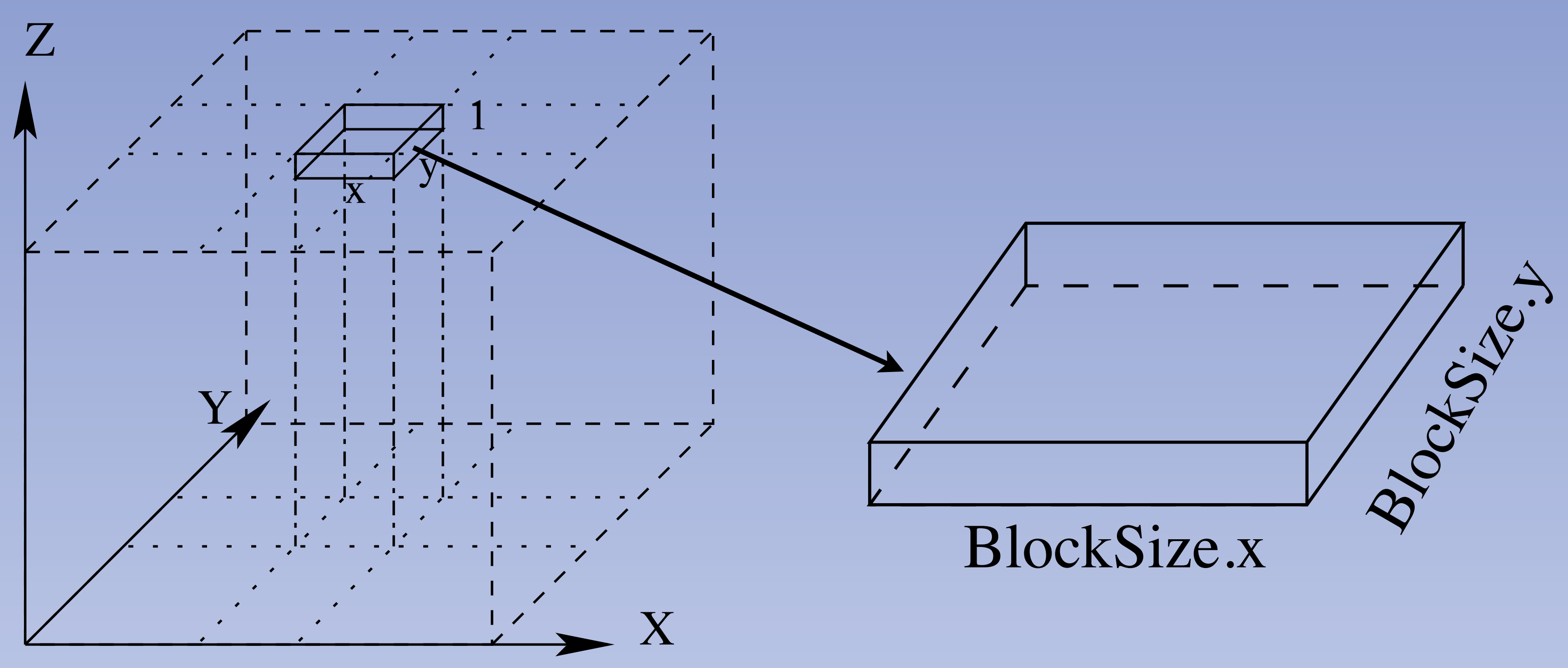
- Parse specification file
- Extract stencil features
  - Halo region (Im, Jm, Km)
  - input/output array name, input scaler name
  - Determine corner or non-corner stencil
  - data type (float or double)

Decomposing Stencil Space



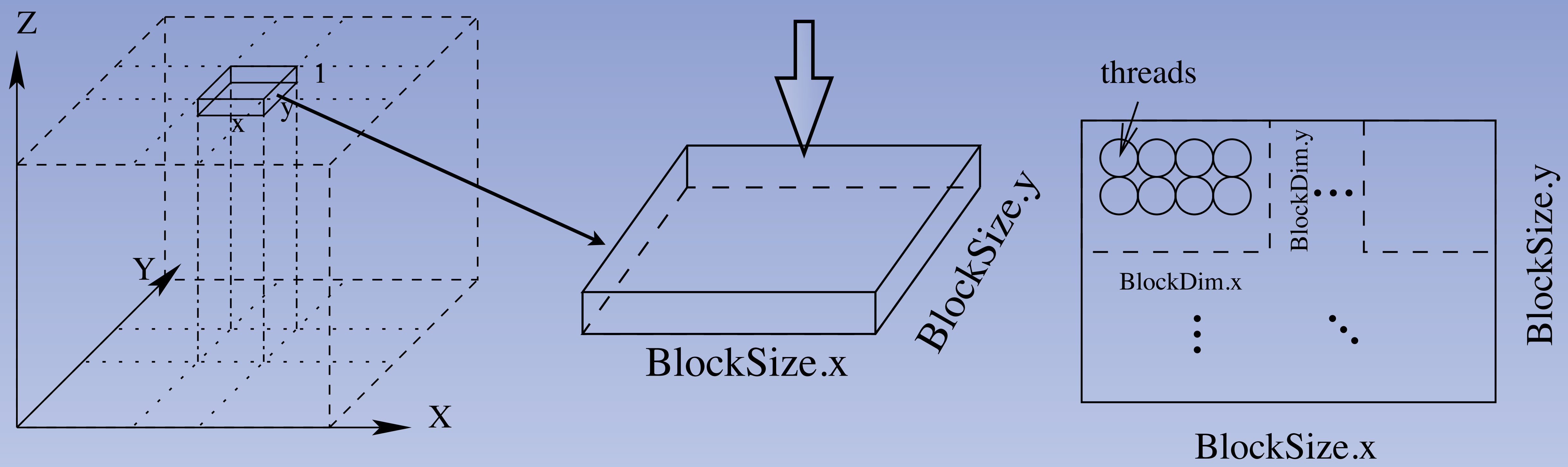
- Tuning parameters
  - BlockSize.x (16, 32, 64)
  - BlockSize.y (2,3..16)
  - BlockDim.x (16, 32, 64, and divisible by BlockSize.x)
  - BlockDim.y (2...16 and divisible by BlockSize.y)
  - Input array texture mapping (boolean)

Decomposing Stencil Space



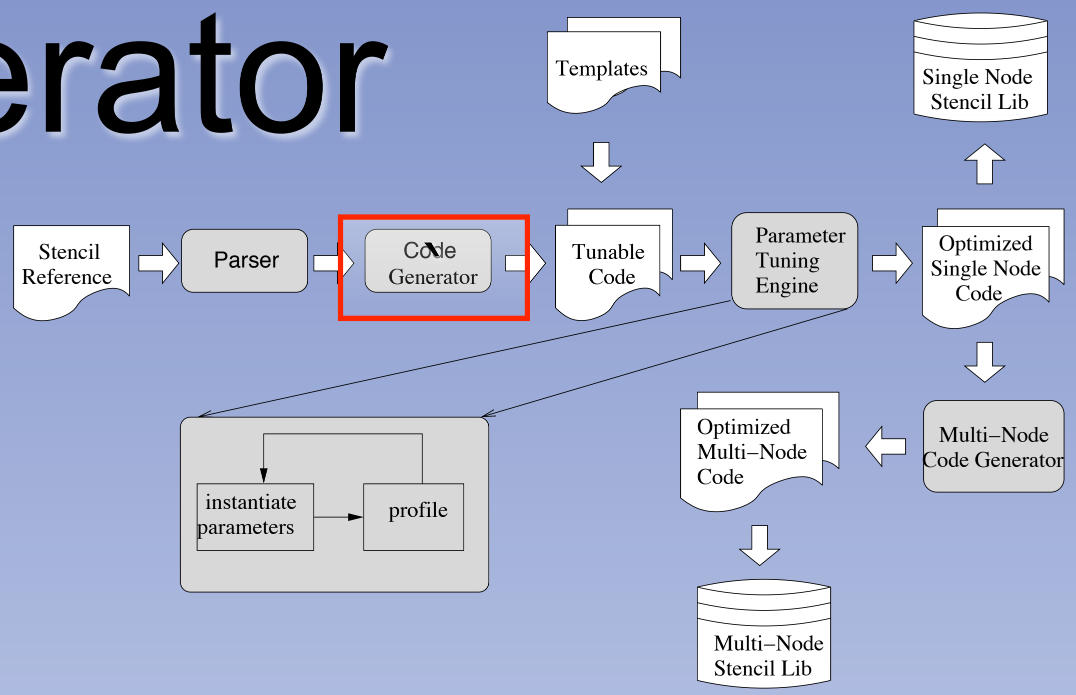
- Tuning parameters
  - BlockSize.x (16, 32, 64)
  - BlockSize.y (2,3..16)
  - BlockDim.x (16, 32, 64, and divisible by BlockSize.x)
  - BlockDim.y (2...16 and divisible by BlockSize.y)
  - Input array texture mapping (boolean)

Decomposing Stencil Space



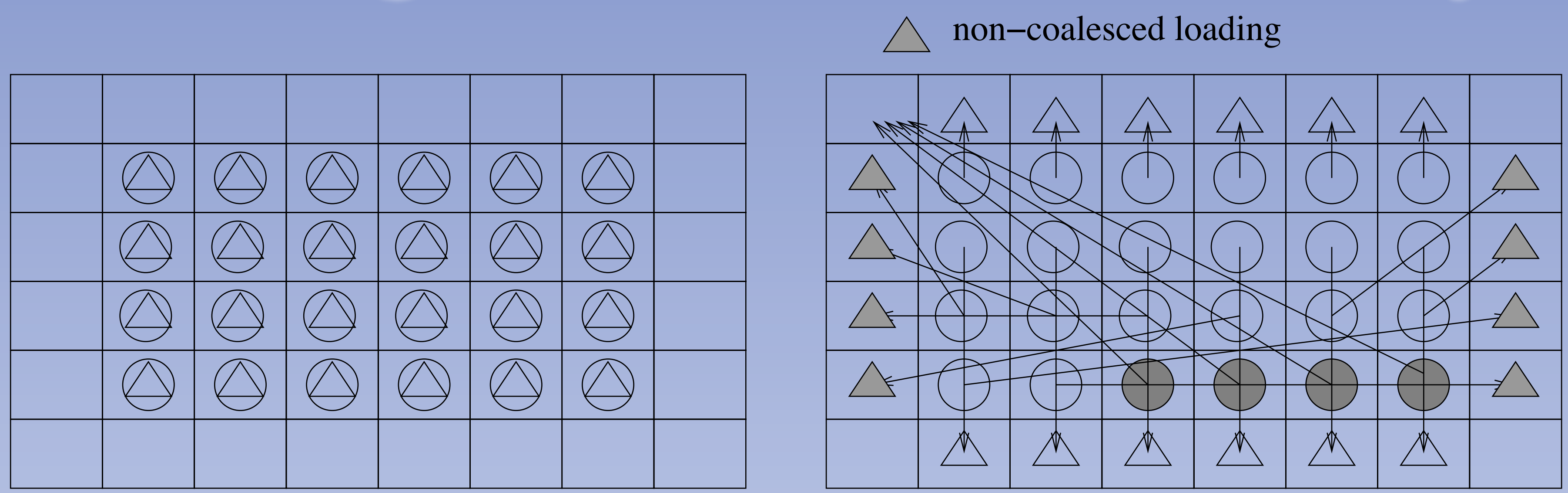
- Tuning parameters
  - BlockSize.x (16, 32, 64)
  - BlockSize.y (2,3..16)
  - BlockDim.x (16, 32, 64, and divisible by BlockSize.x)
  - BlockDim.y (2...16 and divisible by BlockSize.y)
  - Input array texture mapping (boolean)

Code Generator



- Based on two templates
  - With or w/o corner access
  - Different usage of shared memory
    - No corner: Load just one plane to shared memory
    - Corner: Load (1+2*Km) planes to shared memory
    - Experience learned from hand-written stencil code
- Gen unrolling code given block sizes and block dims

Using shared memory

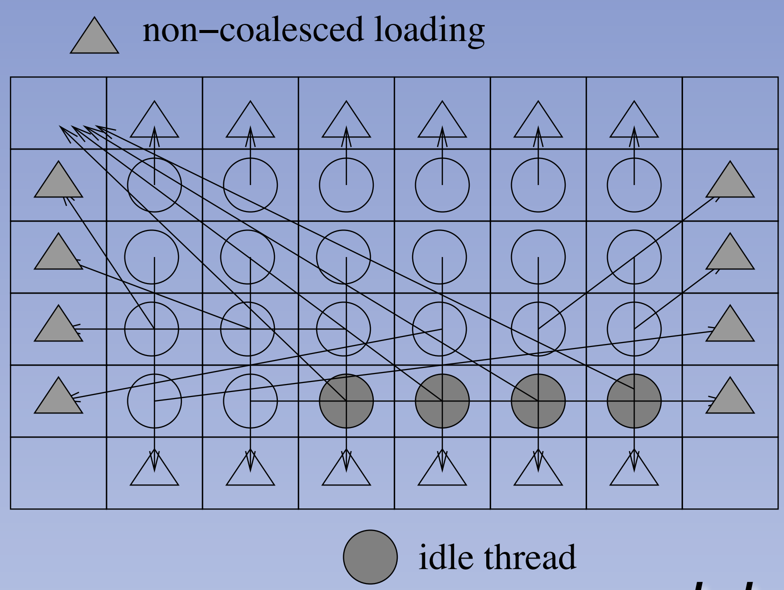


(1)

(2)

- More data points than threads
- One plane at a time
- Load Internal region first (1)
- Code generator to load margins (2)
  - Use constant memory for offset
  - No branches
  - No redundant memory load

Auto-generated Code

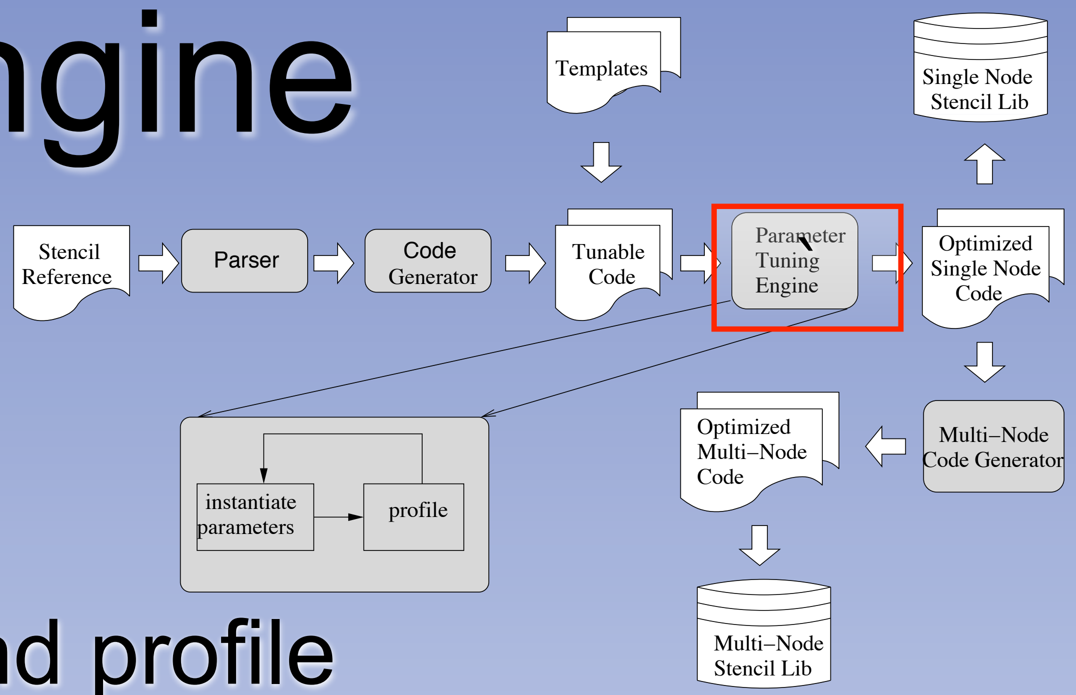



```
__const__ __device__ int haloThreadMappingX[4][6] =
{{1,2,3,4,5,6}, {1,2,3,4,5,6}, {0,0,0,0,7,7}, {7,7,0,0,0,0}};

__const__ __device__ int haloThreadMappingY[4][6]
= {{0,0,0,0,0,0}, {5,5,5,5,5,5}, {1,2,3,4,1,2},
{3,4,0,0,0,0}};

__global__ stencil(...) {
  ...
  halo_offsetx = haloThreadMappingX[threadIdx.x][threadIdx.y];
  halo_offsety = haloThreadMappingY[threadIdx.x][threadIdx.y];
  for (k = HALO_MARGIN_Z; k < zSize + HALO_MARGIN_Z; k++){
     ....
     // load internal
     shMem[threadIdx.y][threadIdx.x] = input[myindexY][myindexX];
     // load margins, no branches
     shMem[halo_offsetx][halo_offsety] = input[myindexY+halo_offsety-
HALO_MARGIN_Y][myindexX+halo_offsetx-HALO_MARGIN_X];
     ...
  }
}
```

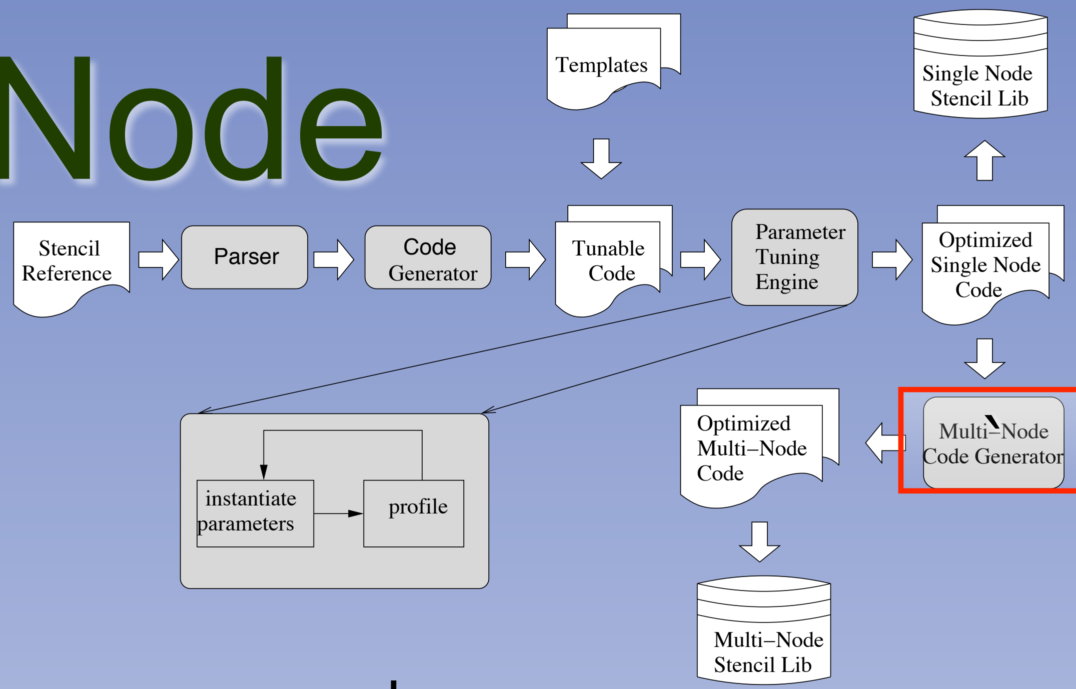
Tuning Engine

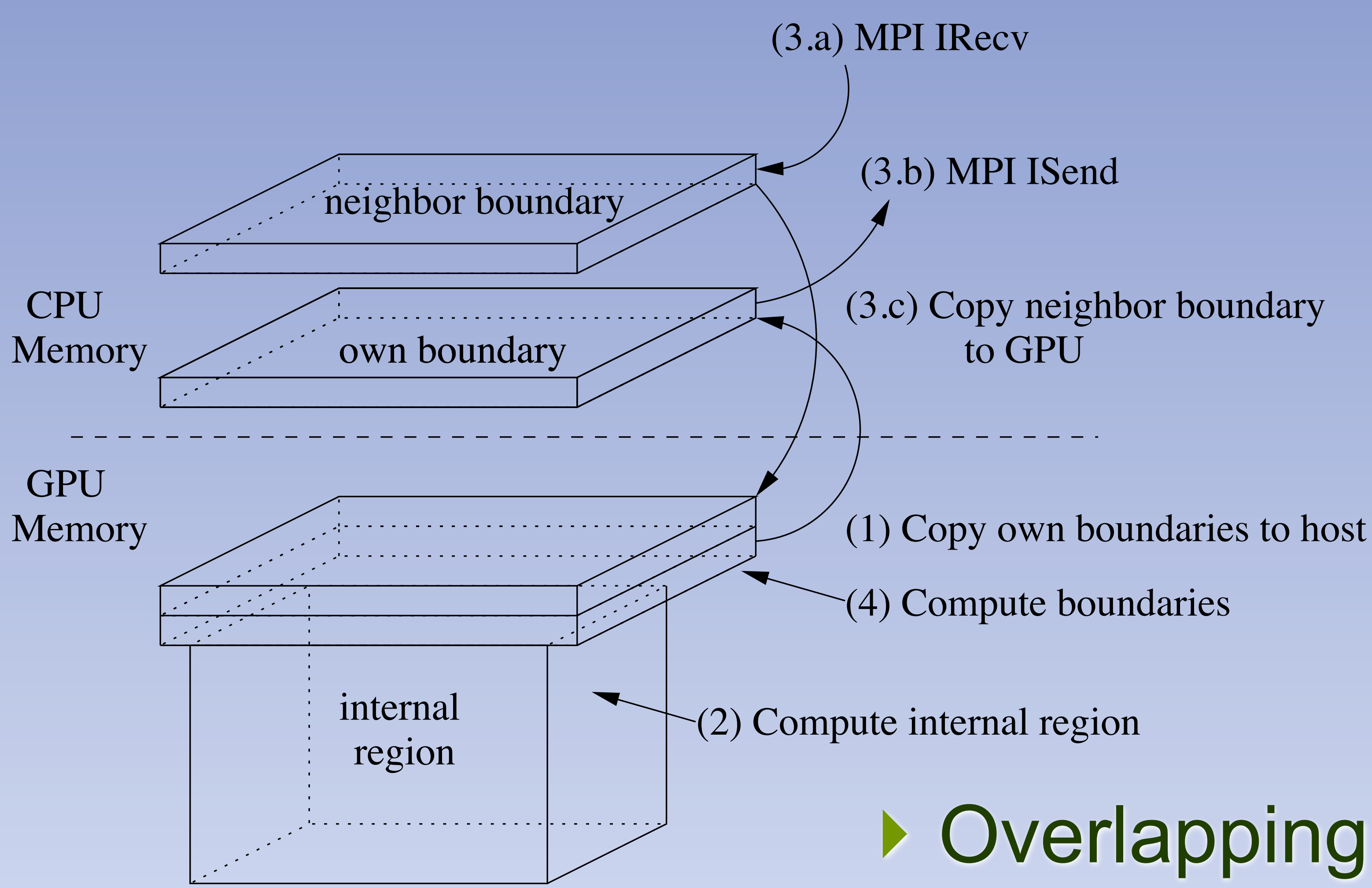


- Brute force search all space
- Instantiate each tuning setting and profile
- Offline

opt = max;
for all block sizes
    for all applicable block dims
        for useTexture = true/false
{
    timing = profile execution;
    opt = timing < opt? timing : opt;
}

Going to Multi-Node





(1)
(2) || (3)
(4)

t

- **Overlapping comp. and comm.**
  - MPI asynchronous send/recv
  - CUDA asynchronous memcpy
- **Separate kernels**
  - For margin layer
  - for non-continuous to continuous memcpy

Experiment Setup

Model	SM Count	Core Count	L1 Cache	Bandwidth(GB/s)	Register	Shared Memory	SP Gflops	DP Gflops
Geforce GTX 280	30	240	N	141.7	16KB	16KB	933	78
Tesla C1060	30	240	N	102.4	16KB	16KB	933	78
Tesla C2050	14	448	Y	144	32KB	16 or 48 KB	1288	515
Geforce GTX 480	15	480	Y	177.4	32KB	16 or 48 KB	1345	168

Single GPU

Cluster	# nodes	Network Bandwidth
Tesla C2050	36	32Gbps
Geforce GTX 480	48	32Gbps

GPU Clusters

Single Node Auto-tuning

Model	BlockSize.x	BlockSize.y	BlockDim.x	BlockDim.y	Texture	SP Gflops
Geforce GTX 280	64/32/64/16	8/8/3/6	32/32/64/16	8/2/3/2	Y/Y/N/N	76.0/117.0/57.6/94.2
Tesla C1060	64/64/64/32	8/6/6/8	32/64/64/32	8/2/3/2	Y/N/Y/N	57.5/91.8/44.8/95.5
Tesla C2050	64/64/64/64	8/6/3/4	32/64/32/32	8/3/3/4	Y/Y/N/Y	87.3/133.8/64.6/157.6
Geforce GTX 480	64/64/64/64	3/3/3/8	32/32/32/32	3/3/3/4	Y/Y/N/Y	108.2/167.8/77.4/203.7

Model	BlockSize.x	BlockSize.y	BlockDim.x	BlockDim.y	Texture	DP Gflops
Geforce GTX 280	16/16/16/16	16/16/16/16	16/16/16/16	4/8/3/3	N/N/Y/N	32.5/35.4/24.0/29.0
Tesla C1060	32/16/32/16	6/16/4/6	32/16/32/16	2/8/2/3	N/n/Y/N	28.8/35.3/22.8/29.3
Tesla C2050	64/32/64/32	8/6/3/6	32/32/64/32	4/2/3/2	Y/Y/N/Y	45.9/66.8/31.8/97.7
Geforce GTX 480	64/32/64/32	6/6/3/4	32/32/64/16	3/2/3/4	Y/Y/N/Y	55.2/77.2/38.7/86.0

7/13/19/27 stencil

No general pattern

Single Node Auto-tuning

Model	BlockSize.x	BlockSize.y	BlockDim.x	BlockDim.y	Texture	SP Gflops
Geforce GTX 280	64/32/64/16	8/8/3/6	32/32/64/16	8/2/3/2	Y/Y/N/N	76.0/117.0/57.6/94.2
Tesla C1060	64/64/64/32	8/6/6/8	32/64/64/32	8/2/3/2	Y/N/Y/N	57.5/91.8/44.8/95.5
Tesla C2050	64/64/64/64	8/6/3/4	32/64/32/32	8/3/3/4	Y/Y/N/Y	87.3/133.8/64.6/157.6
Geforce GTX 480	64/64/64/64	3/3/3/8	32/32/32/32	3/3/3/4	Y/Y/N/Y	108.2/167.8/77.4/203.7

Model	BlockSize.x	BlockSize.y	BlockDim.x	BlockDim.y	Texture	DP Gflops
Geforce GTX 280	16/16/16/16	16/16/16/16	16/16/16/16	4/8/3/3	N/N/Y/N	32.5/35.4/24.0/29.0
Tesla C1060	32/16/32/16	6/16/4/6	32/16/32/16	2/8/2/3	N/n/Y/N	28.8/35.3/22.8/29.3
Tesla C2050	64/32/64/32	8/6/3/6	32/32/64/32	4/2/3/2	Y/Y/N/Y	45.9/66.8/31.8/97.7
Geforce GTX 480	64/32/64/32	6/6/3/4	32/32/64/16	3/2/3/4	Y/Y/N/Y	55.2/77.2/38.7/86.0

7/13/19/27 stencil

No general pattern

Single Node Auto-tuning

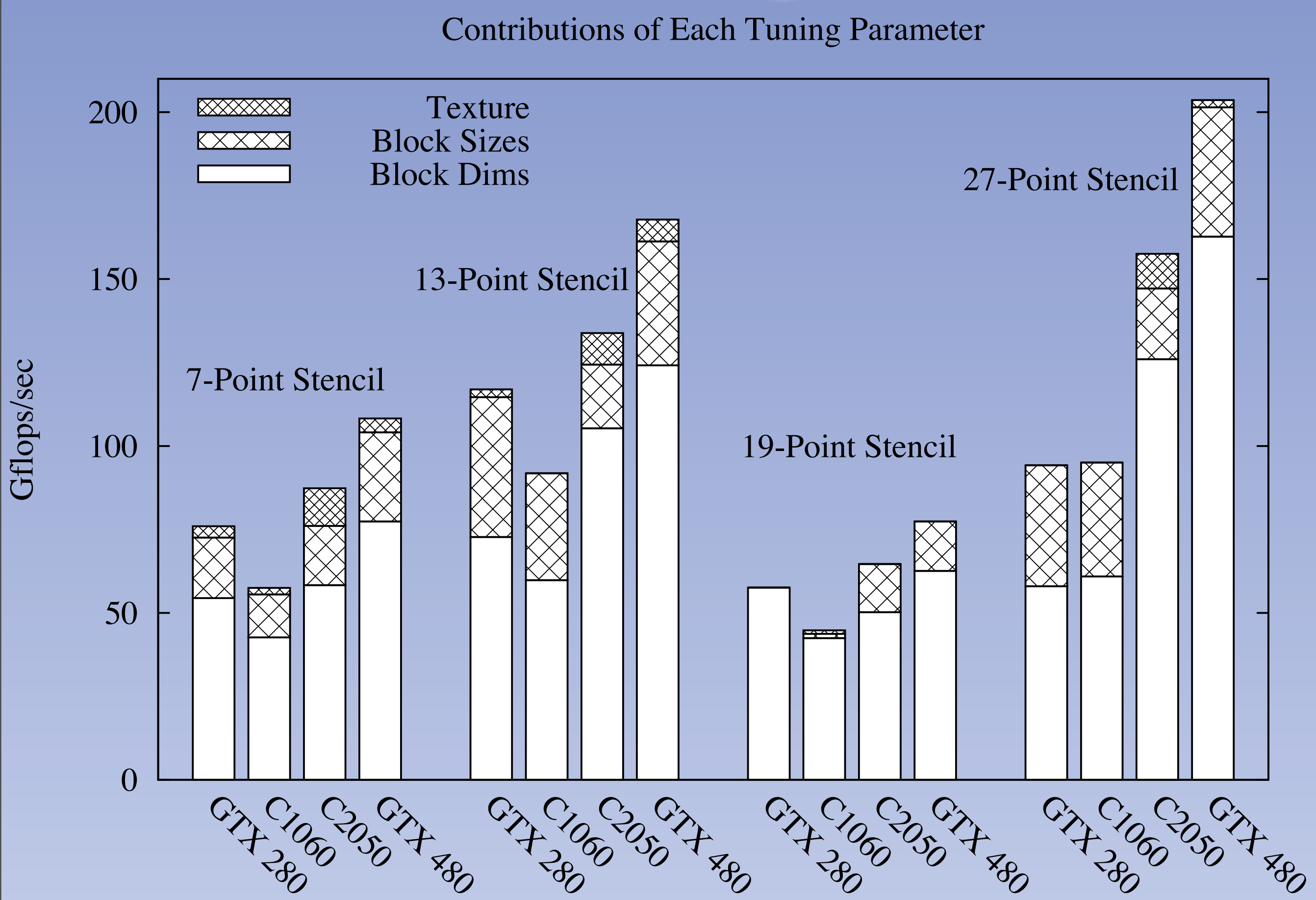
Model	BlockSize.x	BlockSize.y	BlockDim.x	BlockDim.y	Texture	SP Gflops
Geforce GTX 280	64/32/64/16	8/8/3/6	32/32/64/16	8/2/3/2	Y/Y/N/N	76.0/117.0/57.6/94.2
Tesla C1060	64/64/64/32	8/6/6/8	32/64/64/32	8/2/3/2	Y/N/Y/N	57.5/91.8/44.8/95.5
Tesla C2050	64/64/64/64	8/6/3/4	32/64/32/32	8/3/3/4	Y/Y/N/Y	87.3/133.8/64.6/157.6
Geforce GTX 480	64/64/64/64	3/3/3/8	32/32/32/32	3/3/3/4	Y/Y/N/Y	108.2/167.8/77.4/203.7

Model	BlockSize.x	BlockSize.y	BlockDim.x	BlockDim.y	Texture	DP Gflops
Geforce GTX 280	16/16/16/16	16/16/16/16	16/16/16/16	4/8/3/3	N/N/Y/N	32.5/35.4/24.0/29.0
Tesla C1060	32/16/32/16	6/16/4/6	32/16/32/16	2/8/2/3	N/n/Y/N	28.8/35.3/22.8/29.3
Tesla C2050	64/32/64/32	8/6/3/6	32/32/64/32	4/2/3/2	Y/Y/N/Y	45.9/66.8/31.8/97.7
Geforce GTX 480	64/32/64/32	6/6/3/4	32/32/64/16	3/2/3/4	Y/Y/N/Y	55.2/77.2/38.7/86.0

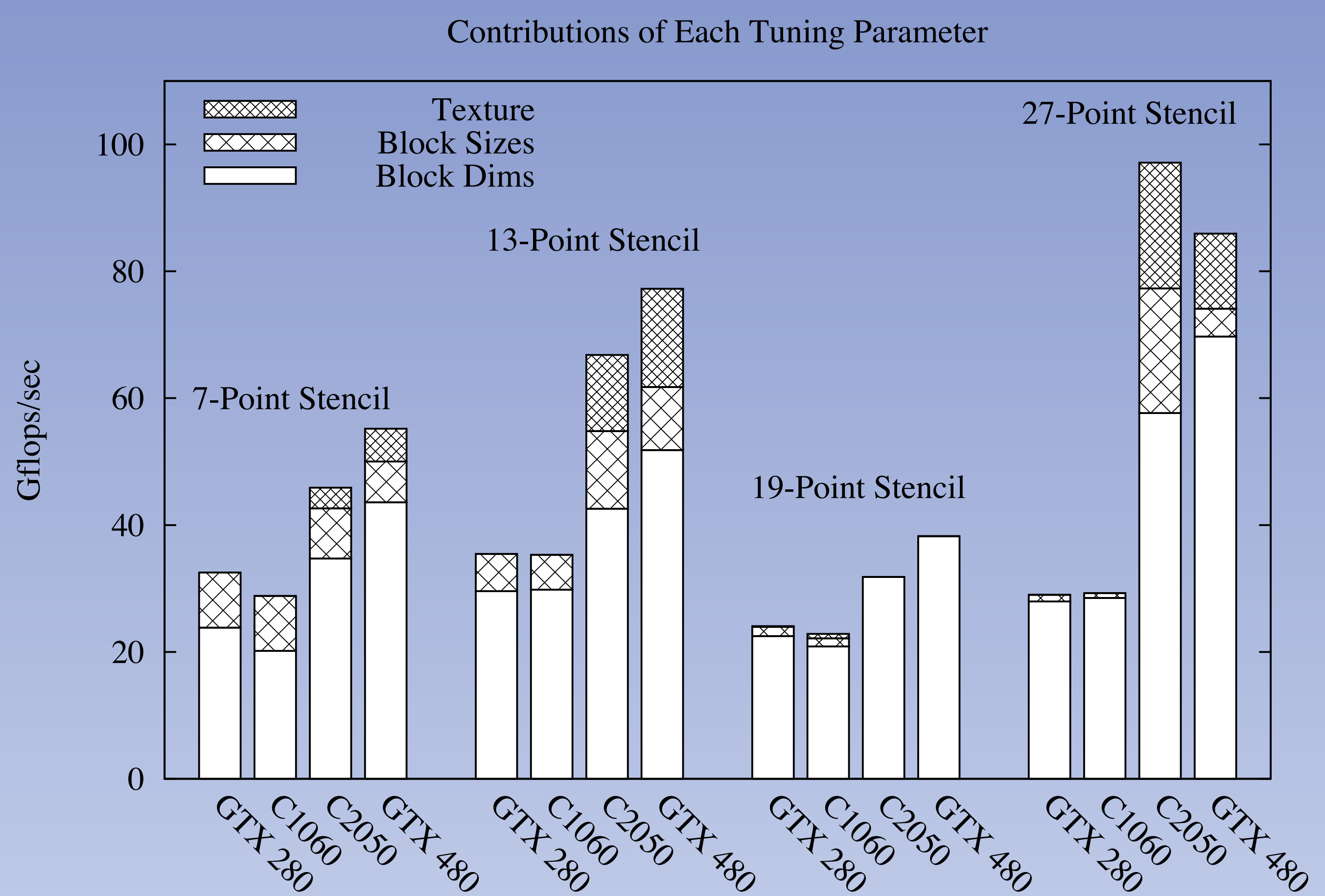
7/13/19/27 stencil

No general pattern

Tuning Effects Breakdowns



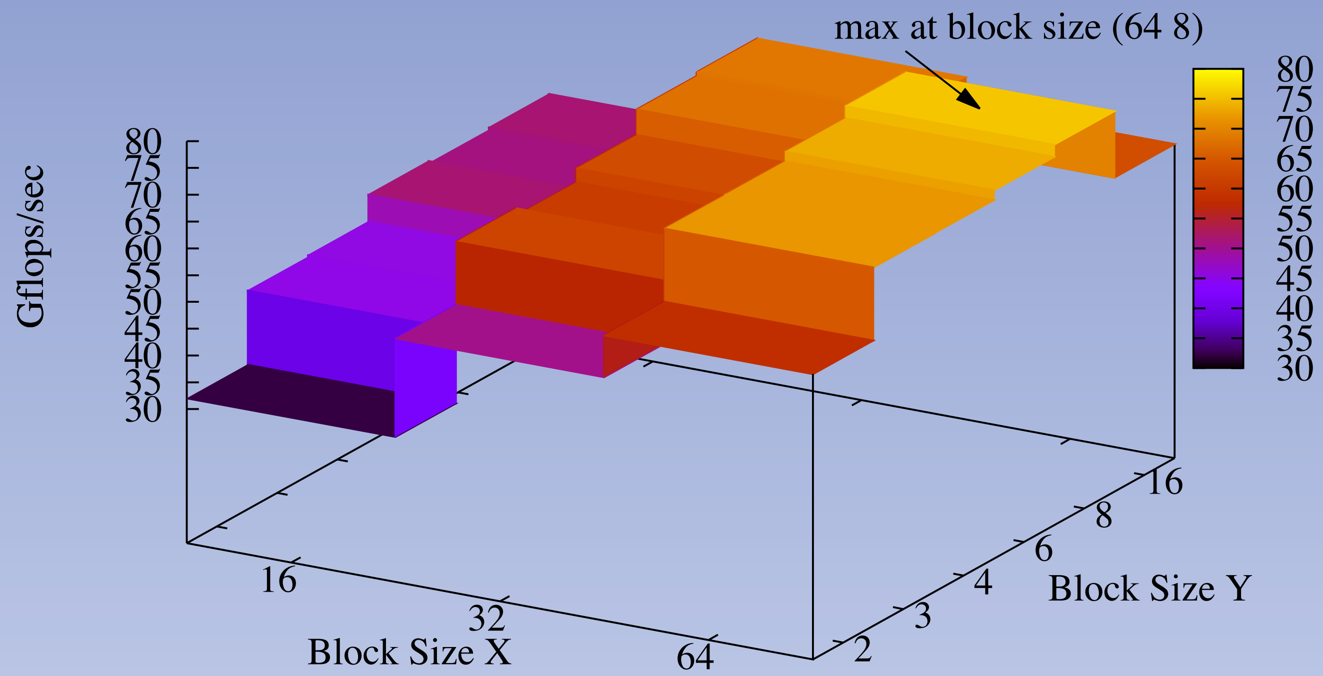
SP stencils



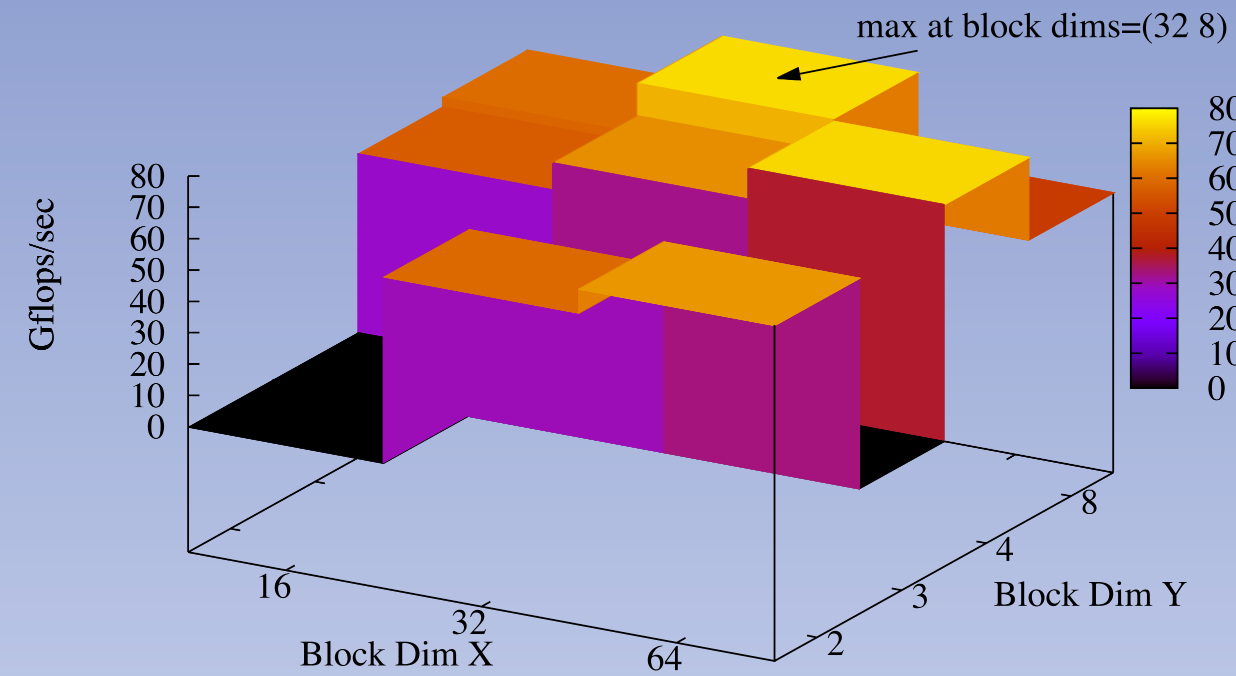
DP Stencils

- *Unrolling is necessary*
  - *BlockDim.x/y almost always differ from BlockSize.x/y*
  - *Except for 19-point DP Stencil for Fermi GPUs (BlockSize.y = 3, too small to unroll)*

Search for Block Sizes and Block Dims



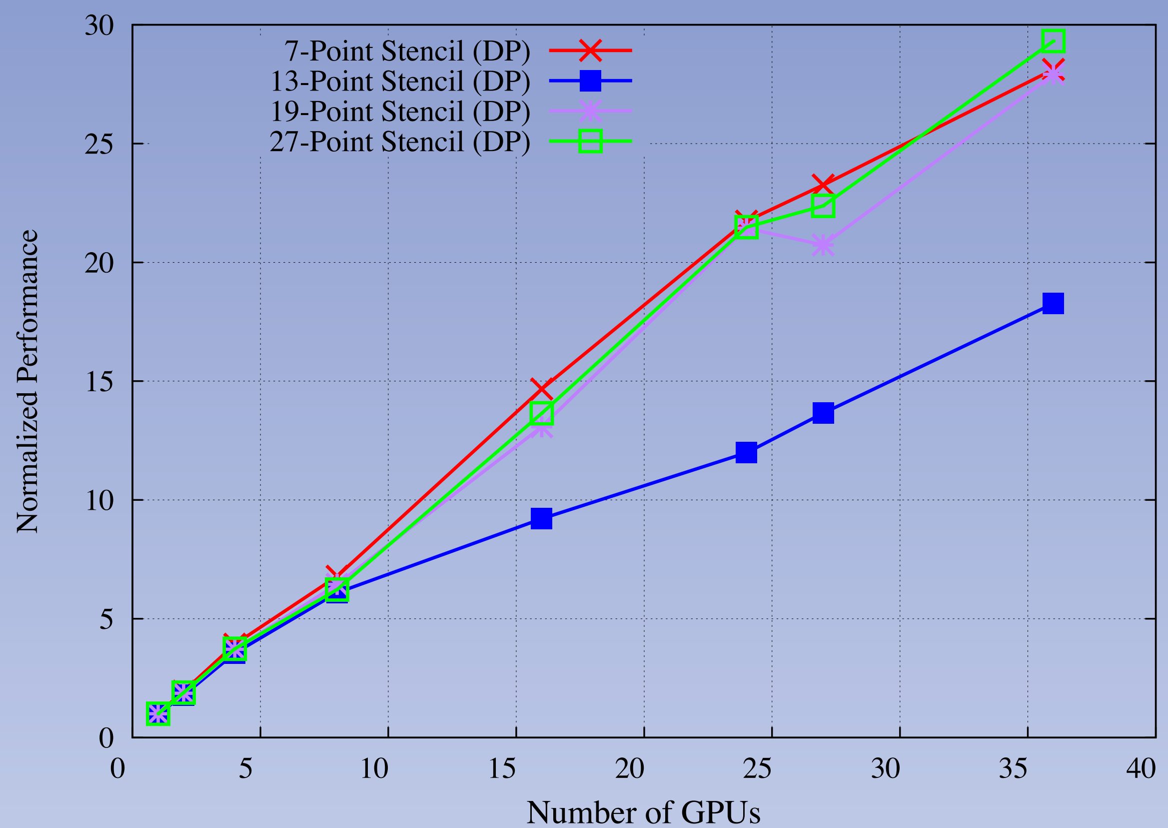
Find Optimal Block Size at (64,8)



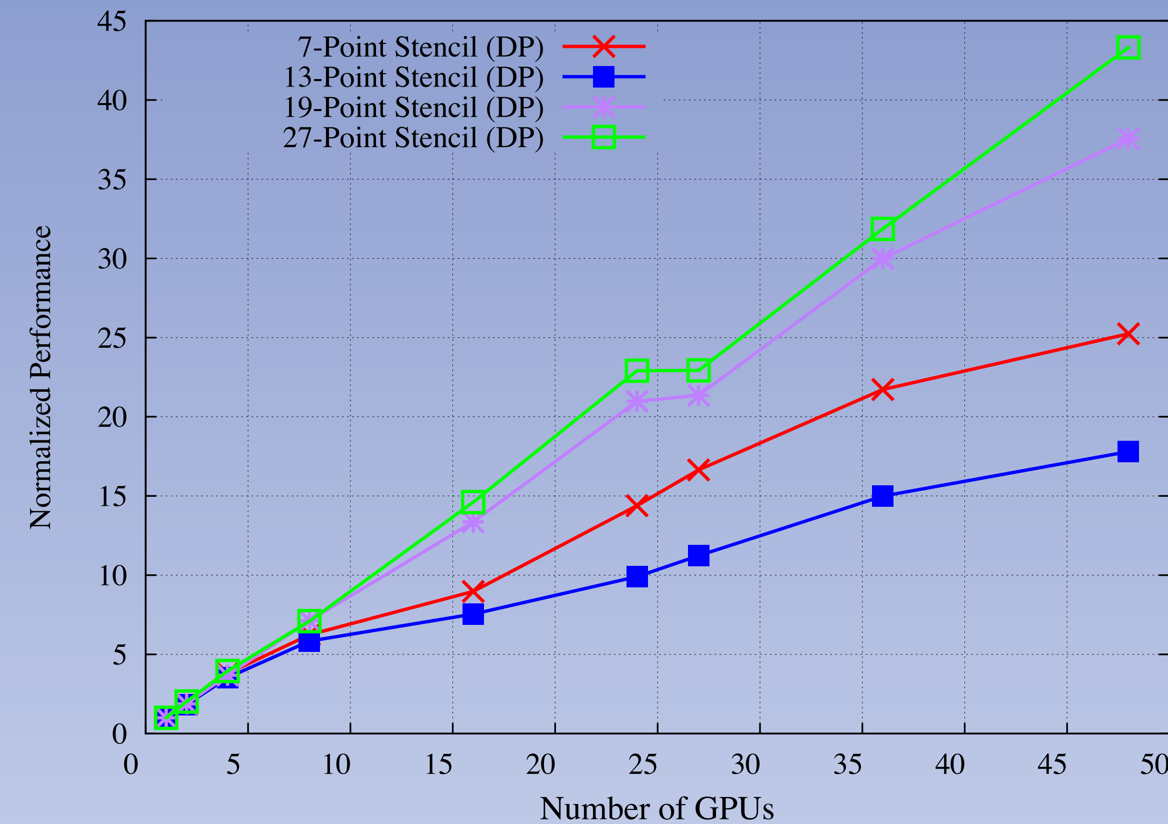
Find Optimal Block Dim, fixing block size at (64,8)

- GTX 280 7-point stencil SP
- Block size first, block dim second
- Local optima exist ((32,8) and (64,4) on right)

Weak Scaling in Multi-Node



C2050 Cluster



GTX 480 Cluster

- 13-point stencil is network bandwidth-bounded
- 7-point stencil comm-bounded in GTX 480 cluster
  - Because of higher Gflops on GTX 480

Comparison with Prior Work

- Our Gflops is 32.5 Gflops for 7-point DP in GTX280
- Datta* (SC'08) 36 Gflops (Same BlockSize and BlockDim)
- Philips* et al. (IPDPS'10, 19-point) 50 Gflops (44.8/47.6(normalized) , similar BlockSize and BlockDim)
- Nguyen* et al. (SC'10) (similar to SC'08)
- Kamil** et al. (IPDPS'10) (14 Gflops)
- Unat** et al. (ICS'11) (22 Gflops on C1060, ours: 28 Gflops)
- Christen** et. al (IPDPS'11) and Maruyama** et al. (SC'11)
  - Only SP results were reported, both inferior to ours.

* Hand written   ** Auto Generation

Comparison with Prior Work

Prior work	Benchmark	Theirs	Ours	Remarks
Datta* (SC'08)	7-point DP on GTX 280	36 GFlops	32.5 GFlops	Same BlockSizes and BlockDims
Philips* et al. (IPDPS'10)	19-point SP on C1060	50 GFlop	44.8/47.6 (normalized)	Similar BlockSizes and BlockDims
Nguyen* et al. (SC'10)	7-point DP on GTX 285	36 GFlops	32.5 GFlops	
Kamil** et al. (IPDPS'10)	7-point DP on GTX 280	14 GFlops	32.5 GFlops	
Unat** et al. (ICS'11)	7-point DP on C1060	22 GFlops	32.5 GFlops	
Christen** et. al (IPDPS'11) al	7-point SP			
Maruyama** (SC'11)	7-point SP			

* Hand written   ** Auto Generation

Conclusion

- GPU programmability and performance not mutually exclusive
- DSL front-end reduces programming effort
- Auto-tuning helps achieving near-optimal performance

Thank you!