Using Modeling to Develop Stencil Codes

Raúl de la Cruz[1] and **Mauricio Araya-Polo**[2]

delacruz@bsc.es, mauricio.araya@shell.com

Barcelona Supercomputing Center (BSC-CNS), Barcelona (Spain)[1]
Shell Intl. E&P Inc., Houston, Texas (USA)[2]

2015 Rice Oil and Gas High Performance Computing Workshop
Houston, Texas, USA, March 4th, 2015

Overview

Stencil in a Nutshell

Finite Difference Method

1: Domain decomposition of mesh
2: **for** *time* = 0 to $time_{end}$ **do**
3: Read Input
4: Pre-processing
5: Inject source
6: Apply boundary conditions
7: **for** all points in my domain **do**
8: Stencil computation (X^t)
9: **end for**
10: Exchange overlapped points
11: Post-processing
12: Write Output
13: **end for**

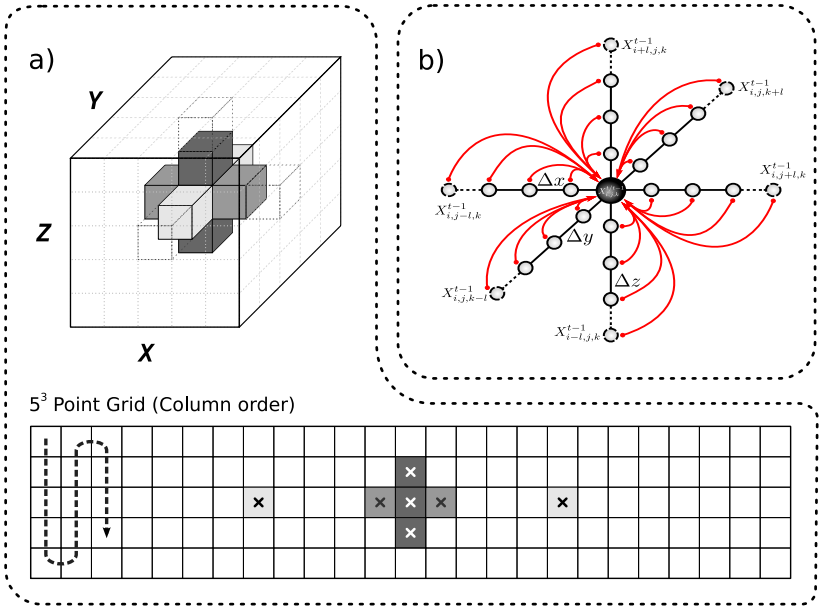
- Load balancing
- Kernel computation
- Intra/inter-node communication

Stencil Computation

for $k = \ell$ to $Y - \ell$ **do**
for $j = \ell$ to $X - \ell$ **do**
for $i = \ell$ to $Z - \ell$ **do**

$$\mathcal{X}^t_{i,j,k} = C_0 * \mathcal{X}^{t-1}_{i,j,k}$$
$$+C_{Z1} * (\mathcal{X}^{t-1}_{i-1,j,k} + \mathcal{X}^{t-1}_{i+1,j,k}) + .. + C_{Z\ell} * (\mathcal{X}$$
$$+C_{X1} * (\mathcal{X}^{t-1}_{i,j-1,k} + \mathcal{X}^{t-1}_{i,j+1,k}) + .. + C_{X\ell} * (\mathcal{X}$$
$$+C_{Y1} * (\mathcal{X}^{t-1}_{i,j,k-1} + \mathcal{X}^{t-1}_{i,j,k+1}) + .. + C_{Y\ell} * (\mathcal{X}$$

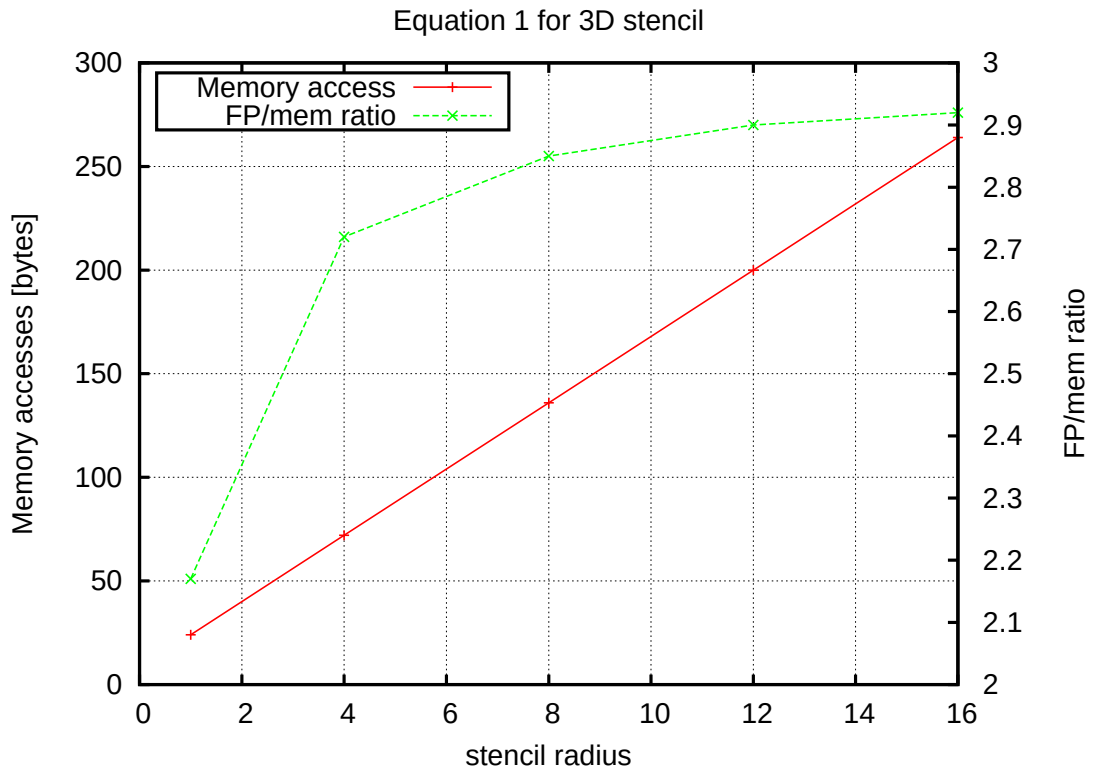
end for
end for
end for



Motivation: Stencil Performance Challenges

Two main challenges:

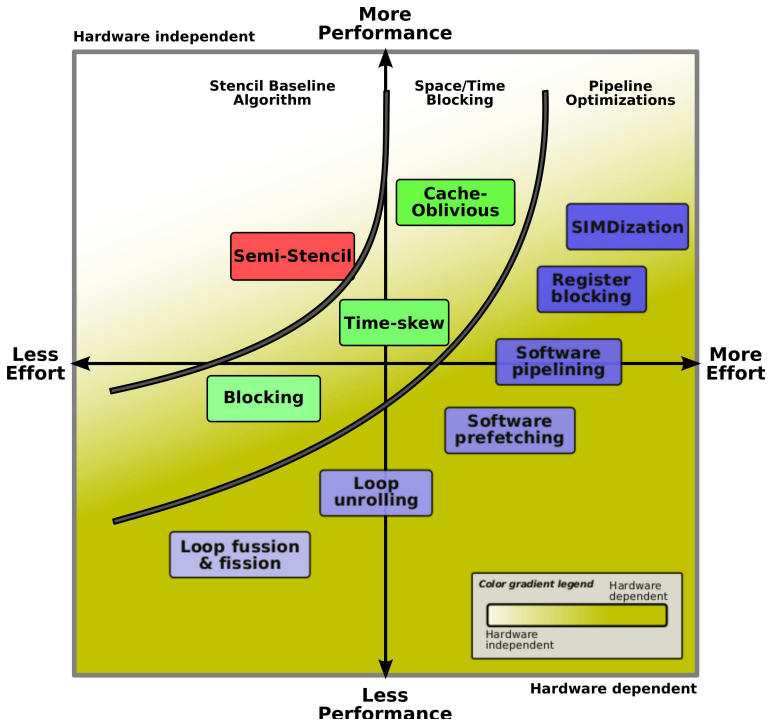
1. Low FLOPs/Memory ratio

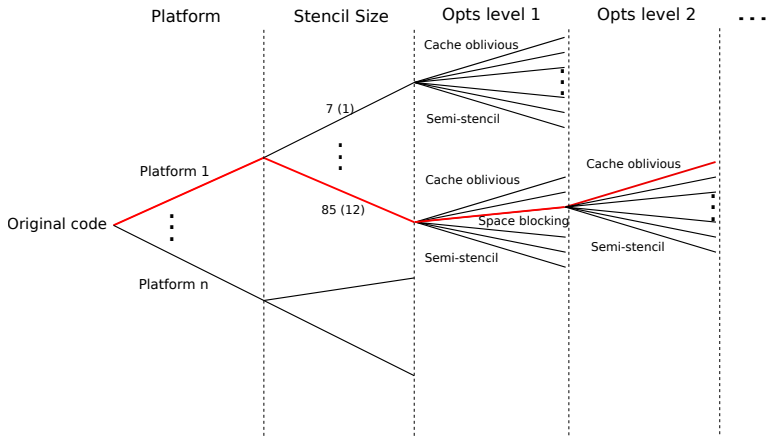


2. Low data reuse

Overview

Approach: Modeling-driven Stencil Code Develop



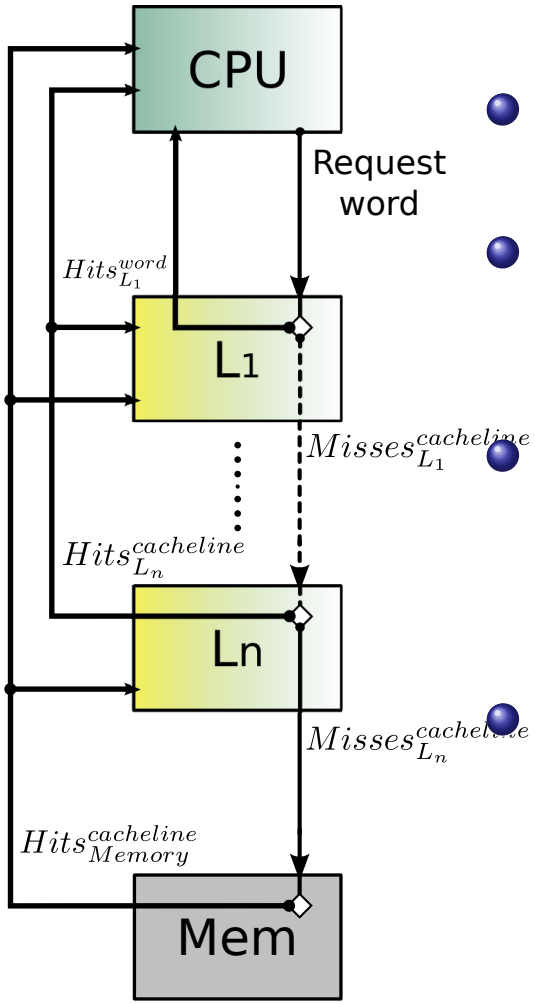


- Which one/combination of these should be implemented?
- How much performance improvement can be expected?
- Brute force (autotuning) is too expensive (combinatorial explosion, code management) then modeling seems to be a valid alternative

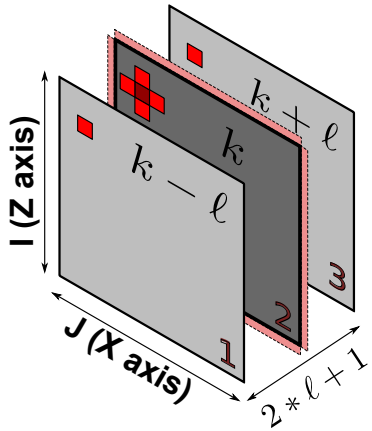
Overview

Stencil Performance Model (I)

Basic considerations:



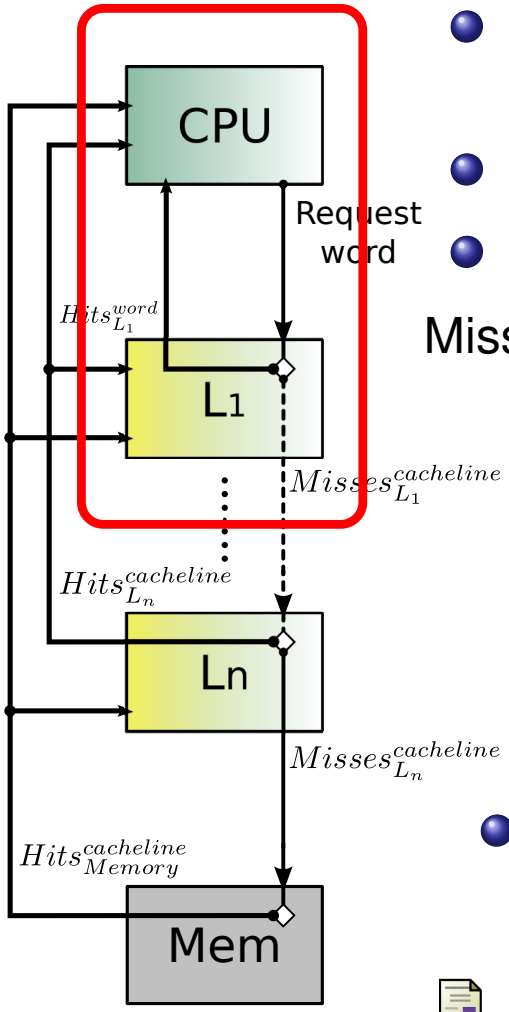
- Computation bottleneck is negligible (low FP/Byte) and reads dominate
- No cache interferences between instructions and data
- P_{read} $(2 * \ell + 1)$ Z-X planes from $\mathcal{X}^{t-1}$ to compute one $\mathcal{X}^{t}$ output plane (P_{write})



- $S_{total} = P_{read} \times S_{read} + P_{write} \times S_{write}$, being $S_{read} = II \times JJ$ and $S_{write} = I \times J$
- Three memory groups are established

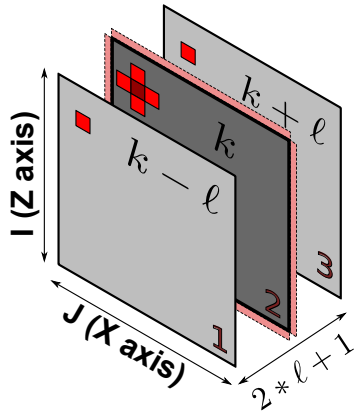
$$T_{total} = \underbrace{T_{L1}}_{first} + \underbrace{\cdots + T_{Li} + \cdots + T_{Ln}}_{intermediate} + \underbrace{T_{Memory}}_{last}$$

Stencil Performance Model (II)



Several data are involved in a stencil comput.:

- Grid points (Input: $\mathcal{X}^{t-1}$, Output: $\mathcal{X}^{t}$)
- Contribution weights ($C_{Z,X,Y,0}$)
- Indices to access grid points (i, j, k)



Misses depend on *II*, *JJ* sizes and ℓ parameter:

$$Misses_{Li}^{cline} = \lceil II/W \rceil * JJ * KK * nplanes_{Li}$$

$$Hits_{Li}^{cline} = Misses_{Li-1}^{cline} - Misses_{Li}^{cline}$$

$$T_{Li}^{cline} = cacheline/Bw_{Li}^{read}$$

$$T_{Li} = Hits_{Li}^{cline} * T_{Li}^{cline}$$

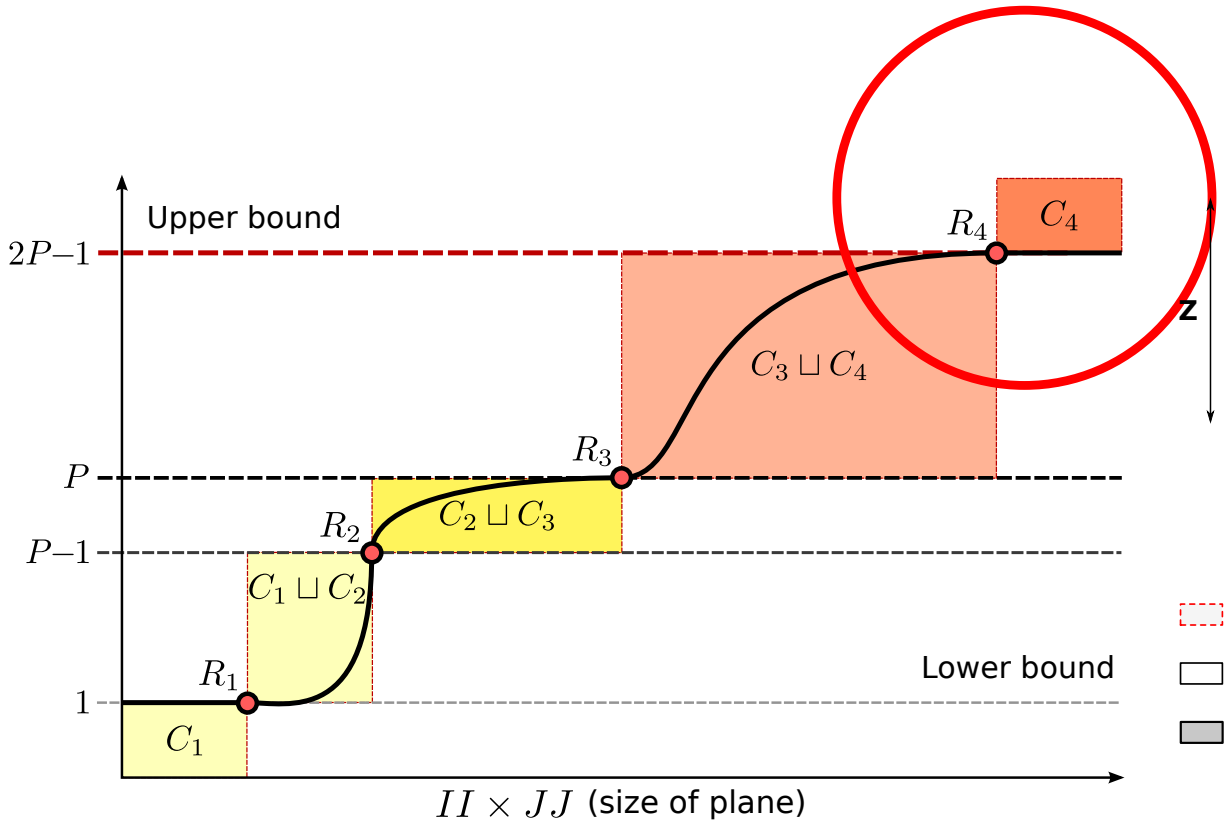
- $nplanes_{Li}$: $II \times JJ$ planes read from $Li + 1$ for each k iteration

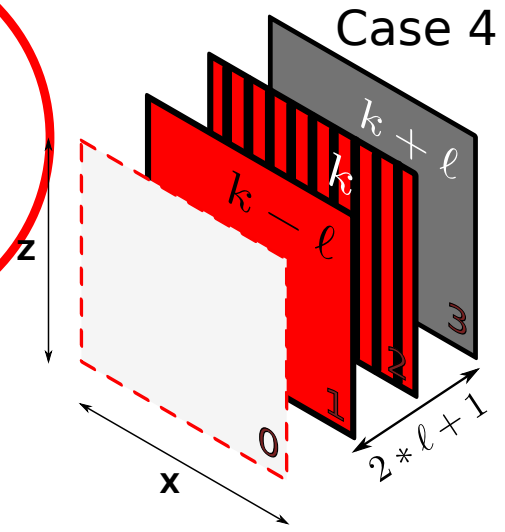
Raúl de la Cruz and Mauricio Araya-Polo.

Towards a Multi-Level Cache Performance Model for 3D Stencil Computation, ICCS11-iWAPT.

Stencil Model (III): Read Miss Cases & Rules

$$nplanes_{Li}(II, JJ) = \begin{cases} C_1 : 1, & \text{if } R_1 \\ C_1 \sqcup C_2 : (1, P_{read} - 1], & \text{if } \neg R_1 \wedge R_2 \\ C_2 \sqcup C_3 : (P_{read} - 1, P_{read}], & \text{if } \neg R_2 \wedge R_3 \\ C_3 \sqcup C_4 : (P_{read}, 2P_{read} - 1], & \text{if } \neg R_3 \wedge \neg R_4 \\ C_4 : 2P_{read} - 1, & \text{if } R_4 \ , \end{cases}$$

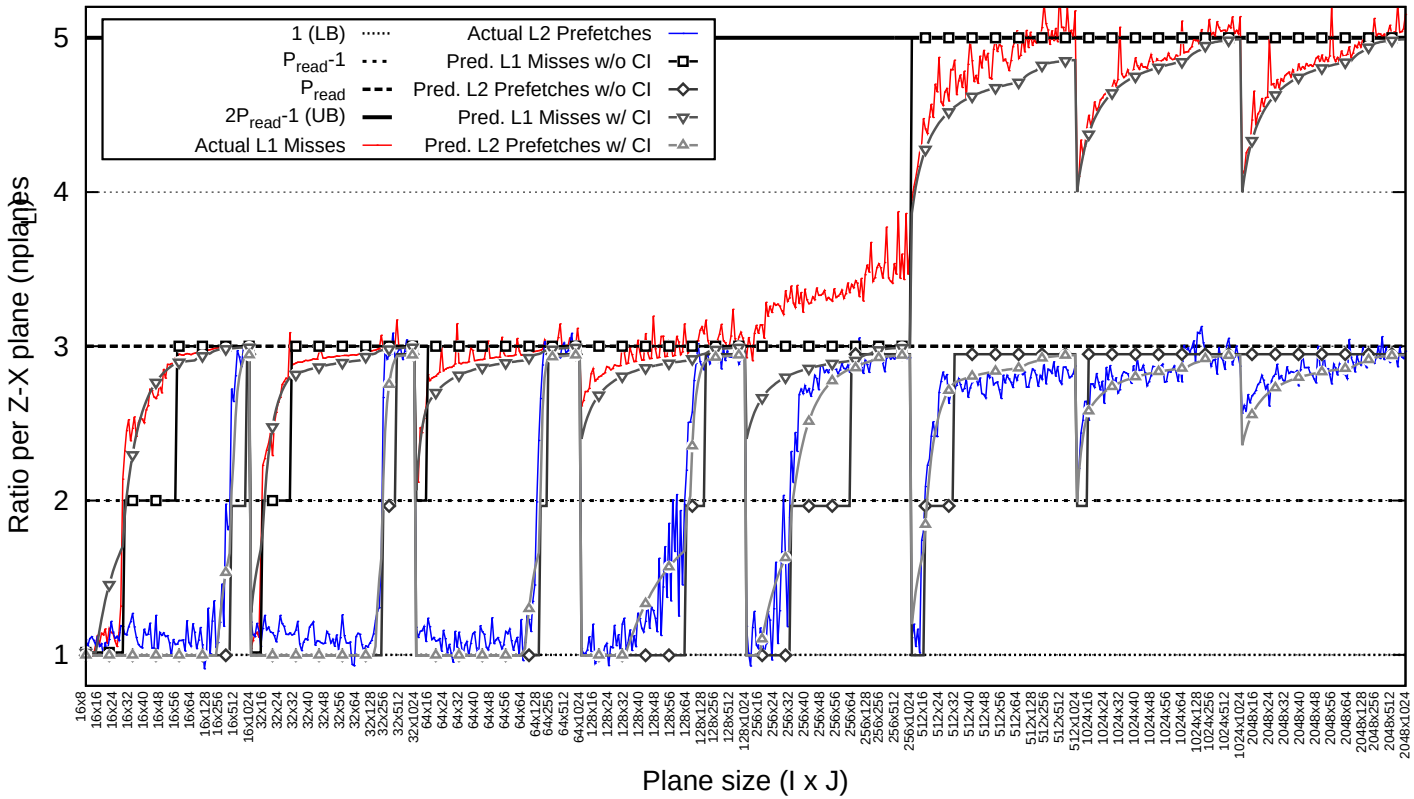




Stencil Model (IV): Cache Interference

- Add full 3C (compulsory, conflict and capacity) misses detection
- Convert the model from a discrete to a continuum space

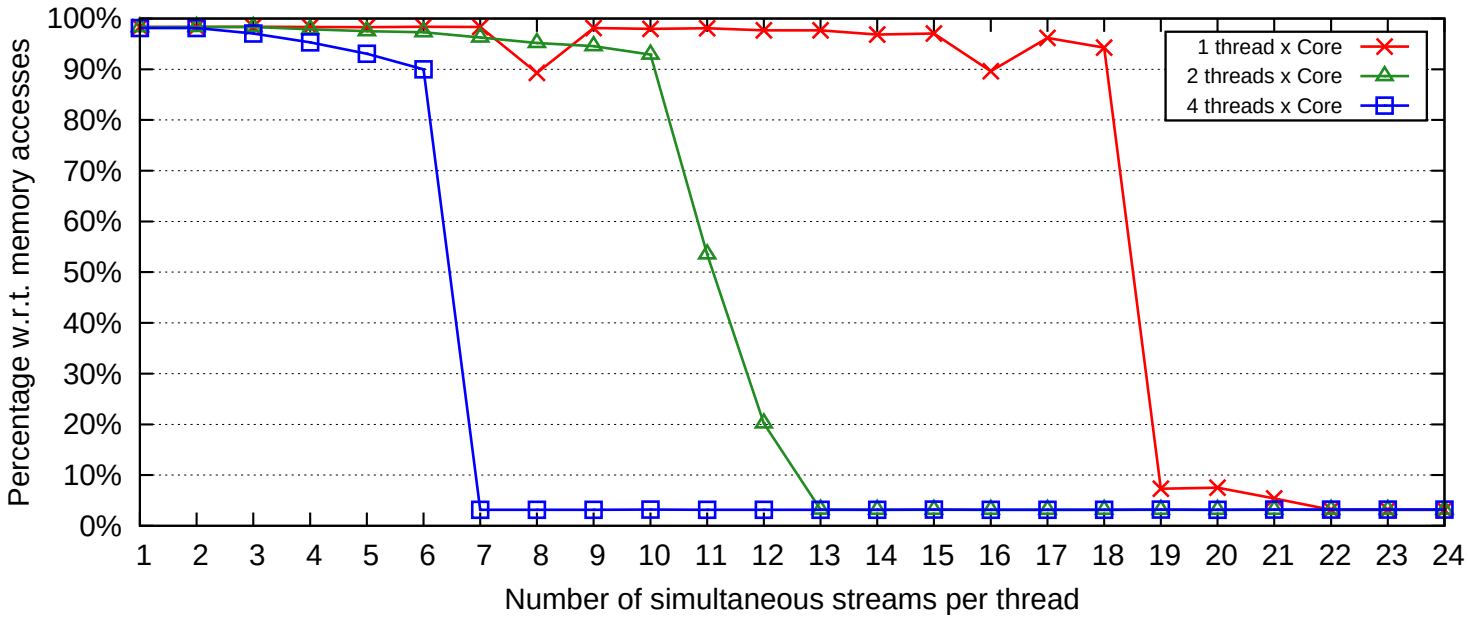
$$P(i) = \frac{\overbrace{II \times JJ}^{\text{whole plane}} - \overbrace{II \times (P_{read} - 1)}^{\text{columns reuse}}}{II \times JJ} \in [0,1] \rightarrow nplanes_{Li}\prime = nplanes_{Li} \times P(i)$$



Stencil Model (V): Prefetching Effect

Gabriel Marin, Collin McCurdy, and Jeffrey S. Vetter.
Diagnosis and optimization of application prefetching performance. (ICS '13)

$$DC_{effectiveness} = DC\_Req\_PF / DC\_Req\_All \in [0, 1]$$



$$nplanes_{Li}^{S} = nplanes_{Li} \times DC_{effectiveness}$$

$$nplanes_{Li}^{NS} = nplanes_{Li} \times (1 - DC_{effectiveness})$$

Overview

Stencil in a Nutshell

Modeling-driven Stencil Code Development

Stencil Performance Model
Model
Cache Interference
Prefetching Approach

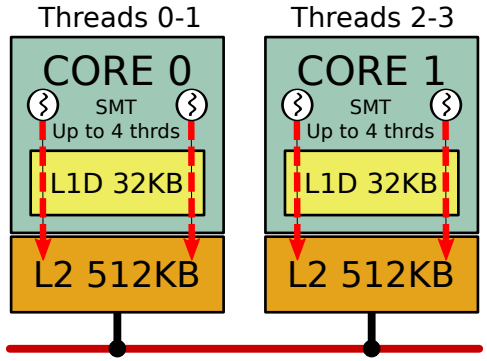
Model Validation
Cache Miss Prediction
Core Efficiency in SMT Mode

How to use the model

Conclusions & Future Work

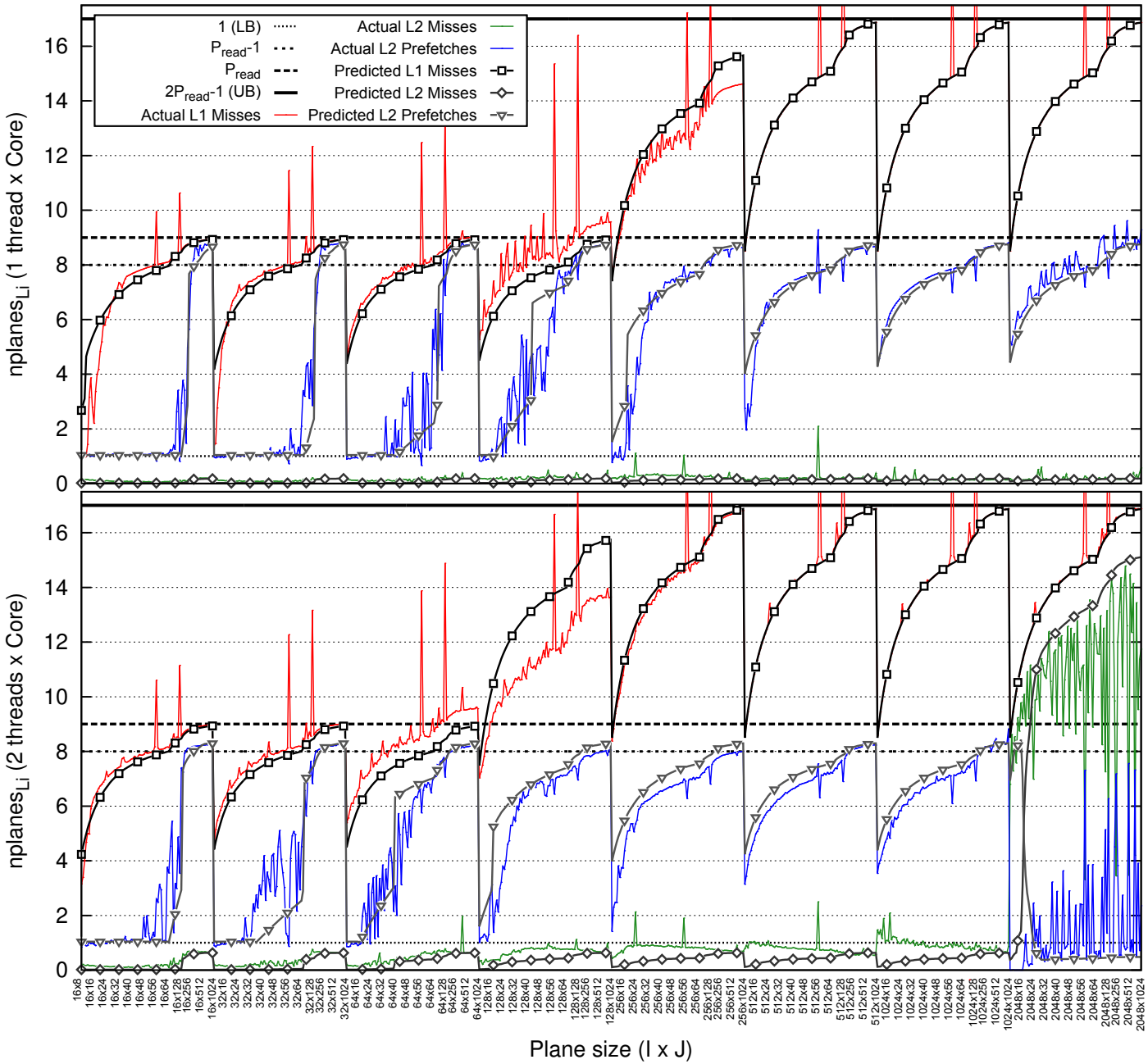
Model Validation: Experiments on Intel Xeon Phi

Parameters	Range of values
Naive sizes ($I \times J \times K$) Rivera sizes ($I \times J \times K$) Stencil sizes (ℓ)	$8 \times 8 \times 128 \ldots 2048 \times 1024 \times 128$ $512 \times 2048 \times 128$ 1, 2, 4 and 7 (7, 13, 25 and 43-point)
Algorithms	{Naive, Rivera} $\times$ {Classical, *Semi-stencil*}
Block sizes (*TI* and *TJ*)	{8, 16, 24, 32, 64, 128, 256, 512, 1024, 1536, 2048}
SMT configuration	1, 2 and 4 threads x Core



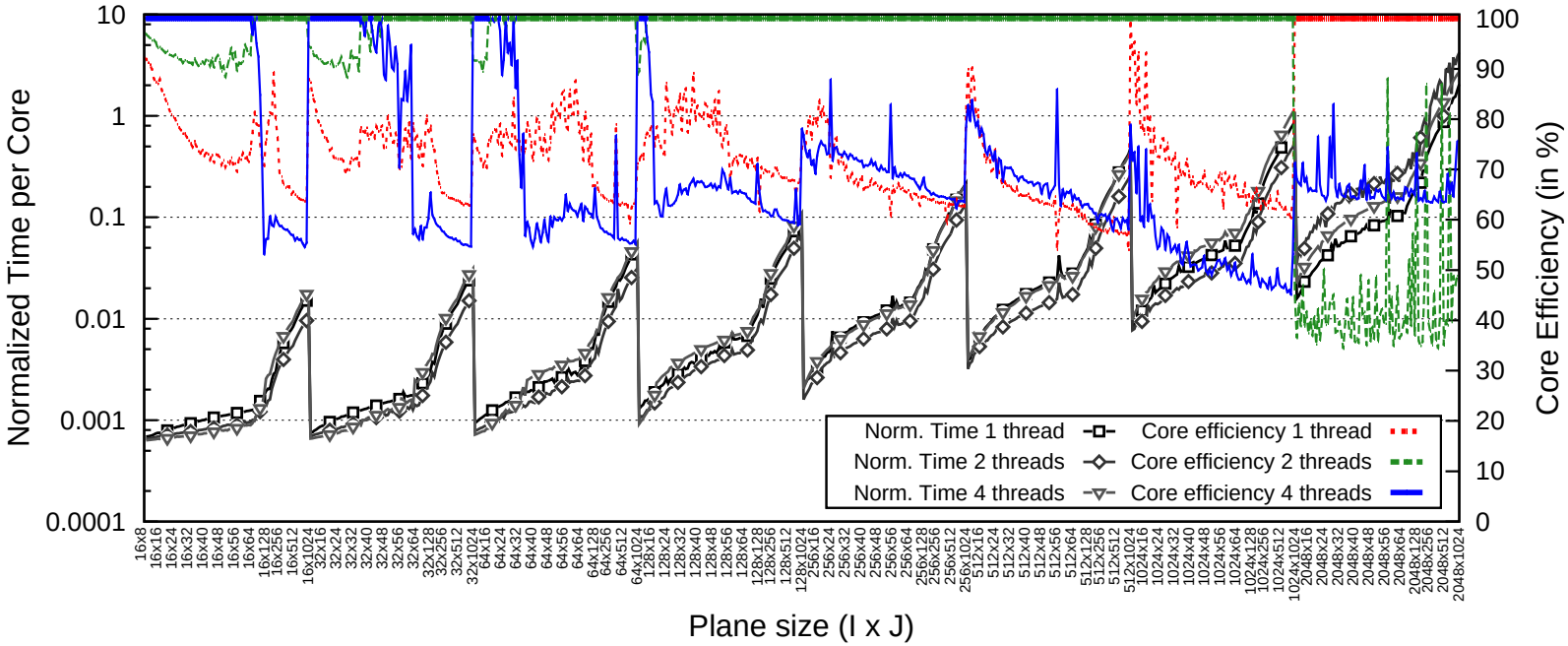
Description	Intel Xeon Phi Events	Time Cost Formulas
Cycles	CPU_CLK_UNHALTED	$T_{L1} = (\text{L1 Reads - L1 Misses}) \times Bw_{L1}^{cli}$
L1 Reads	VPU_DATA_READ	$T_{L2} = Bw_{L2}^{cline} \times (\text{L1 Misses - L2 Misse}$
L1 Misses	VPU_DATA_READ_MISS	$(\text{L2 Prefetches - L2 Writes} \times \text{Pref}$
L2 Misses	L2_DATA_READ_MISS_ MEM_FILL	$T_{Mem} = \text{L2 Misses} \times Bw_{Mem}^{NS} + Bw_{Mem}^{S}$ $(\text{L2 Prefetches - L2 Writes} \times \text{Pref}$
L2 Prefetches	HWP_L2MISS	$T_{Writes} = \text{L2 Writes} \times \text{Pref Eff} \times Bw_{Wri}^{S}$
L2 Writes	L2_WRITE_HIT	$+ \text{L2 Writes} \times (1 - \text{Pref Eff}) \times Bw_{W}^{N}$

Model Validation (I): Cache Misses (Naive case, $\ell = 4$)



Model Validation (II): Core Efficiency in SMT Mode

$$Core^{SMT_i}_{efficiency} = \frac{\min(\tau^{SMT_1}, \ldots, \tau^{SMT_n})}{\tau^{SMT_i}} \in [0, 1]$$



- Decide best SMT configuration to be conducted
- Predict cache and prefetching contention (due to ℓ and $II \times JJ$)

Raúl de la Cruz and Mauricio Araya-Polo.
Modeling Stencil Computations on Modern HPC Architectures, SC14-PMBS.

Developing Stencil Code Using The Model

1. Get the model (soon in github)
2. Run stream benchmark or similar
3. Edit platforms.h (HW parameters) using results from step 2
4. Compile for any supported architecture (SANDY, KNC, NEHALEM, IVY, OPTERON, SHANGHAI, POWER6, BGP, PPC970)
5. Set the problem size *IxJxK*, block size (if blocking will be used), stencil order and time iterations
6. Run model (e.g):
   $>$ model.1thr.1xCore.LIN_INTERP 64 64 64 16 16 16 20 8
7. The output can help:
   - quick and cheap estimation of performance,
   - to decide which combinations of order/blocking sizes and threads per code to use (KNC case), before even developing any code;
   - evaluate different optimizations performance,
   - if after step 2 a roofline model was created, results of our model can be compared against roofline

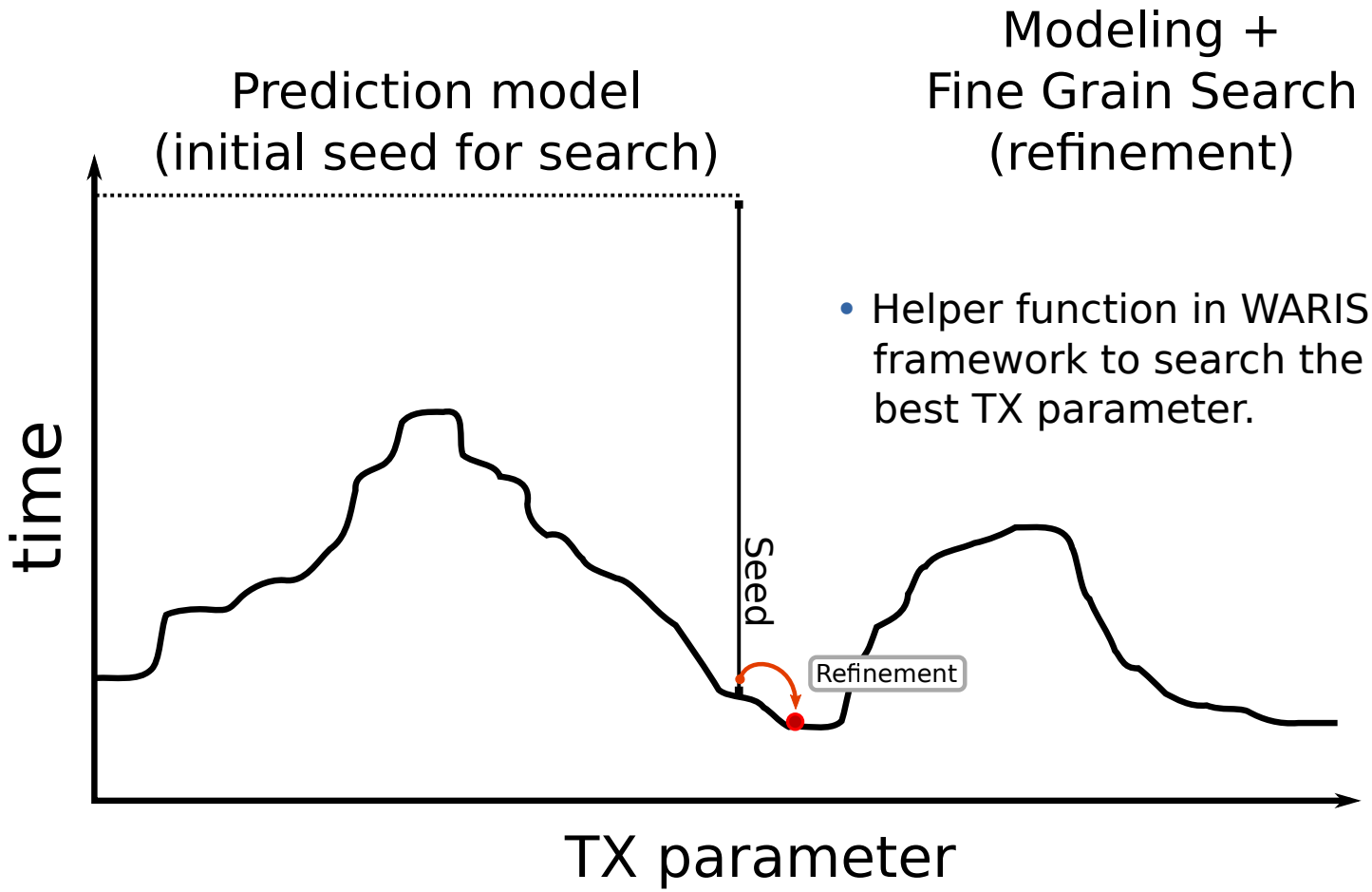
Conclusions & Future Work

- This model includes multi- and many-core support
- Robust hardware prefetching modeling (*prefetching effectiveness*)
- Considers cache interference phenomena (3C misses)
- Includes stencil optimizations (blocking and semi, more coming)
- Results are accurate (rel. error 1-15%)
- Core efficiency predictor (best SMT configuration)
- Useful for static and dynamic analysis for making development and budget decisions

Q&A

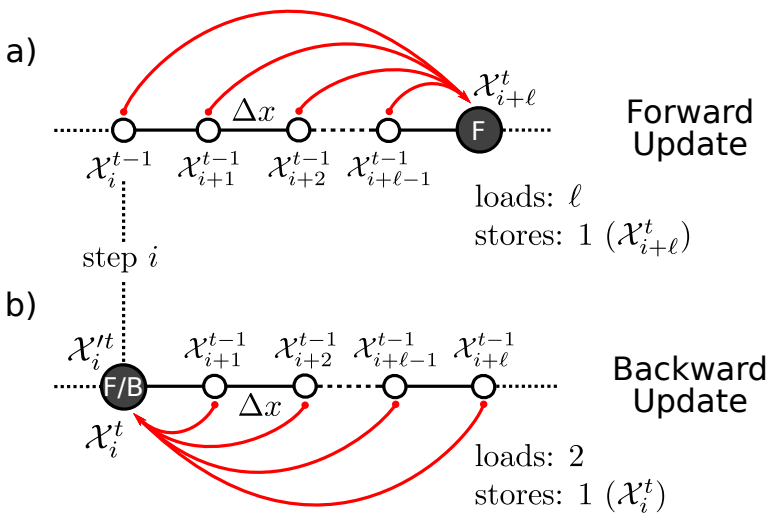
Thanks!

Auto-tuning blocking parameters

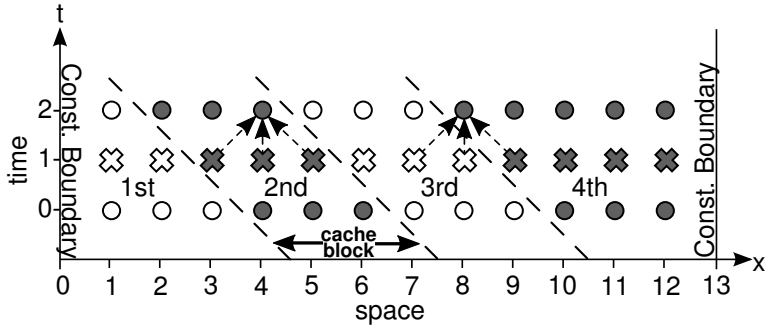


- Helper function in WARIS framework to search the best TX parameter.

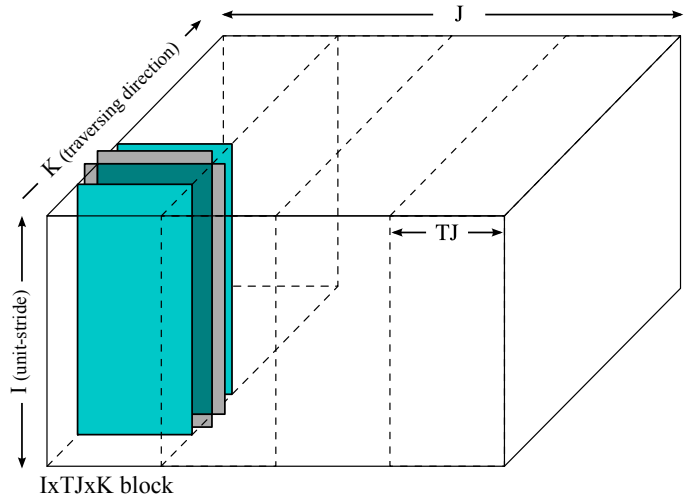
State of the Art - The most popular



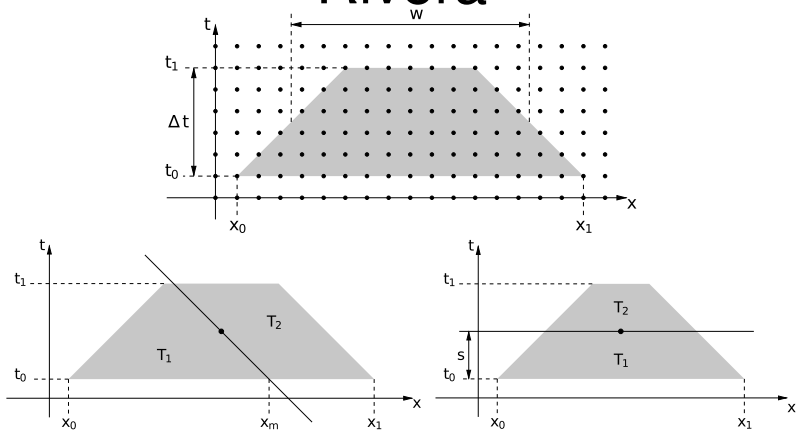
Semi-stencil



Time-skewing



Rivera



Cache-oblivious

Stencil Model: Read Miss Cases & Rules