Stencil Computations: from Academy to Industry

Raúl de la Cruz, Mauricio Hanzich and Jose María Cela

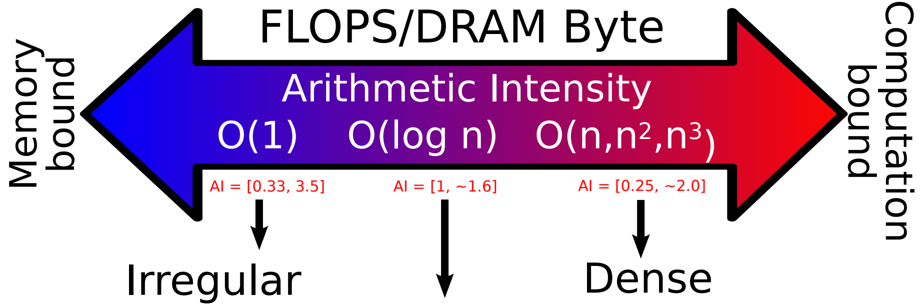
{raul.delacruz,mauricio.hanzich,josem.cela}@bsc.es

Barcelona Supercomputing Center, Barcelona (Spain)

Parallel Processing Conference
Optimizing Stencil-based Algorithms
Portland, Oregon, USA, February 18-21, 2014

FLOPS are cheap - MEMORY access is costly

Optimization is a **MUST**



Irregular codes (Stencils & SpMV)

FFT

FFTW

Dense algebra (BLAS-like DGEMM)

ALYA **WARIS**

ATLAS **ESSL**

Stencil in a Nutshell

Finite Difference Method

```
Domain decomposition of mesh
for time = 0 to time_end do
    Read Input
    Pre-processing
    Inject source
    Apply boundary conditions
    for all points in my domain do
        Stencil computation (X^t)
    end for
    Exchange overlapped points
    Post-processing
    Write Output
end for
```

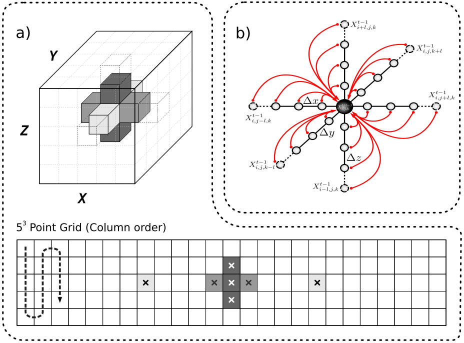
- Load balancing
- Kernel computation
- Intra/inter-node communication

Stencil Computation

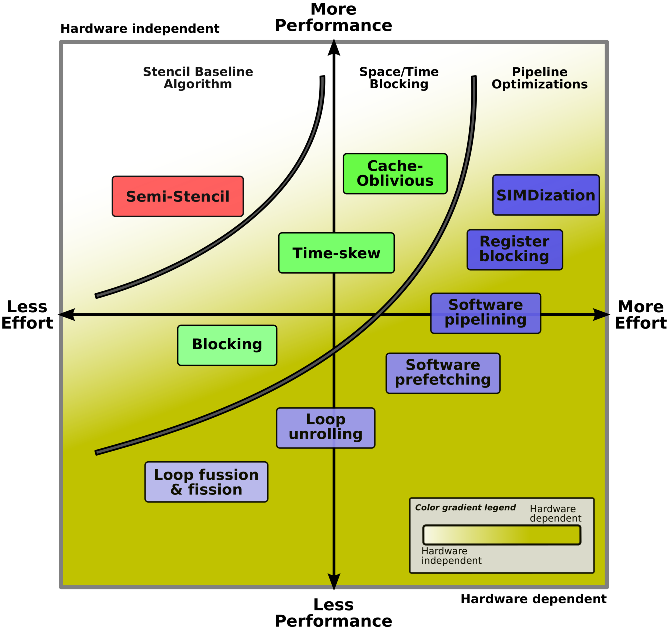
for $k = \ell$ to $Y - \ell$ **do**
  for $j = \ell$ to $X - \ell$ **do**
    for $i = \ell$ to $Z - \ell$ **do**

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

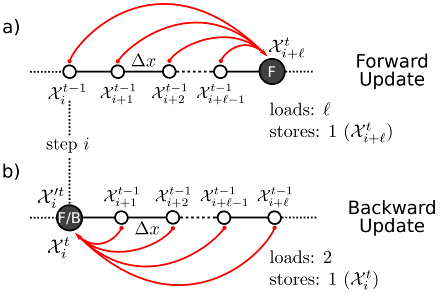
    end for
  end for
end for



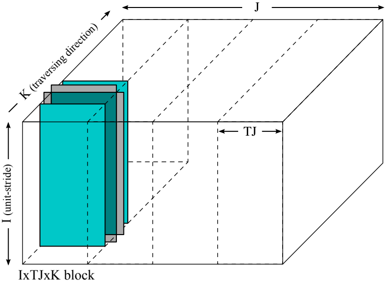
State of the Art - The Big Picture



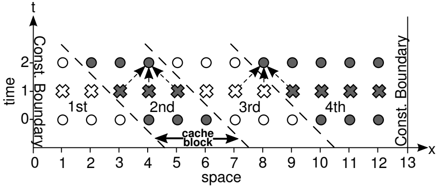
State of the Art - The most popular



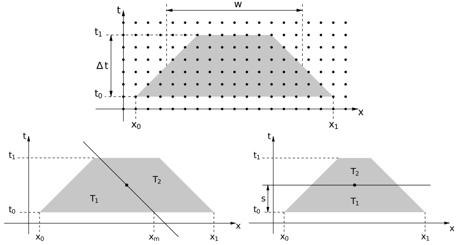
Semi-stencil



Rivera

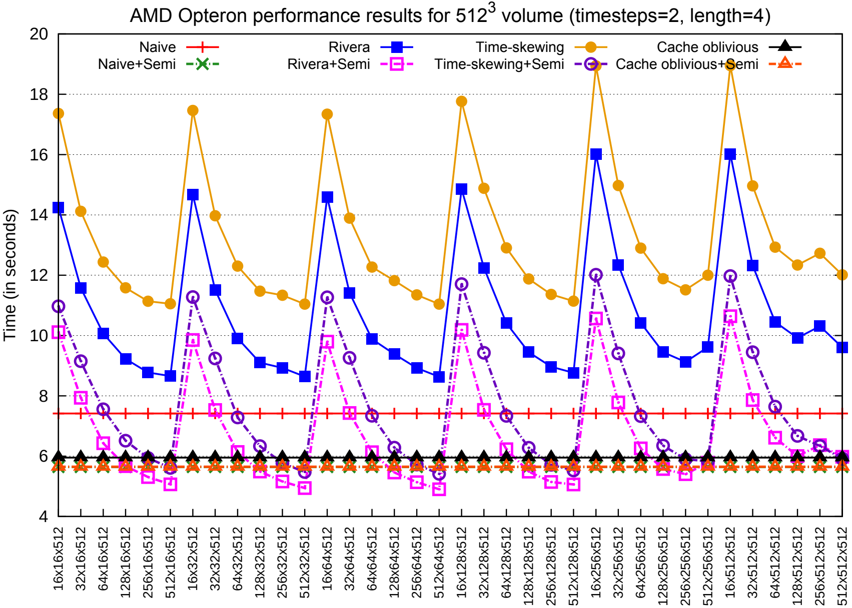


Time-skewing



Cache-oblivious

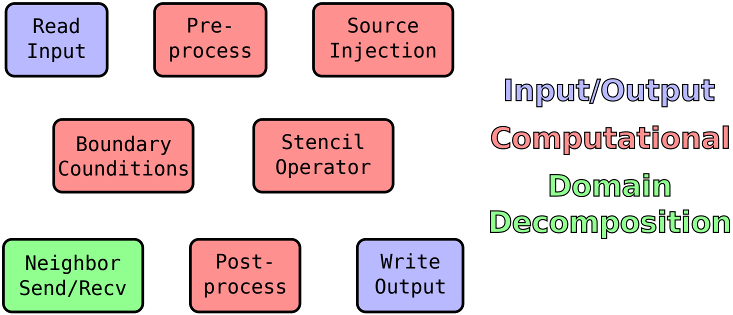
State of the Art - Crunching numbers



The reality

The black boxes

- Many stages are involved in an Industrial FD scheme code
- Stencil computation is the core of PDE+FD based simulations (up to 80% of execution time)
- However, the optimization of the remaining stages are also crucial in order to boost simulations

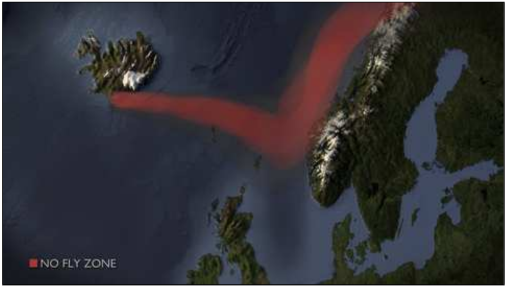


Ash Transport-Deposition model

Advection-Diffusion-Sedimentation eq.:

- Concentration (C^t,C^{t+1})
- Wind velocity (U_x,U_y,U_z)
- Diffusion (K_x,K_y,K_z)
- Source (S_*) Air density (ρ_*)

$$\frac{\partial C}{\partial t} + \frac{\partial}{\partial X}(U_x C) + \frac{\partial}{\partial Y}(U_y C) + \frac{\partial}{\partial Z}[(U_z - V_{sj})C] =$$
$$\frac{\partial}{\partial X}\left(\rho_* K_x \frac{\partial C/\rho_*}{\partial X}\right) + \frac{\partial}{\partial Y}\left(\rho_* K_y \frac{\partial C/\rho_*}{\partial Y}\right) +$$
$$\frac{\partial}{\partial Z}\left(\rho_* K_z \frac{\partial C/\rho_*}{\partial Z}\right) + S_*$$



Read Input

Meteorological data
- U, V, W, rho, T
- p, u*, zi, L
- Interpolate data

Pre-process

Preprocess meteorological data:
- Compute diffusion terms (K_x, K_y, K_z)
- Compute Vsettling and Vdrydeposition
- Scale terms, from LON-LAT to Cartesian
- Scale precipitation rates (mm s^{-1} to mm h-1)
- Add radial wind
- Compute div(u) (ensure mass-conservation law)
- Precompute fields: Vsj=Uz-Vsj, [Kx,Ky,Kz]*rho

Source Injection

Source injection:
- Ash particles ejected by volcano

Boundary Counditions

Boundary Conditions (P1 and P2 stages):
- Zero derivative for outgoing flux
- Null concentration for incoming flux

Stencil Operator

Stencil computation (P1 and P2 stages):
- Advection term: Lax-wendorff scheme 2n-order (space/time), with slope limiter (minmod)
- Diffusion: central difference scheme

Neighbor Send/Recv

Communicate P1:
- MPI_Send/Recv (extra-domains)
- Shared memory (intra-domains)

Post-process

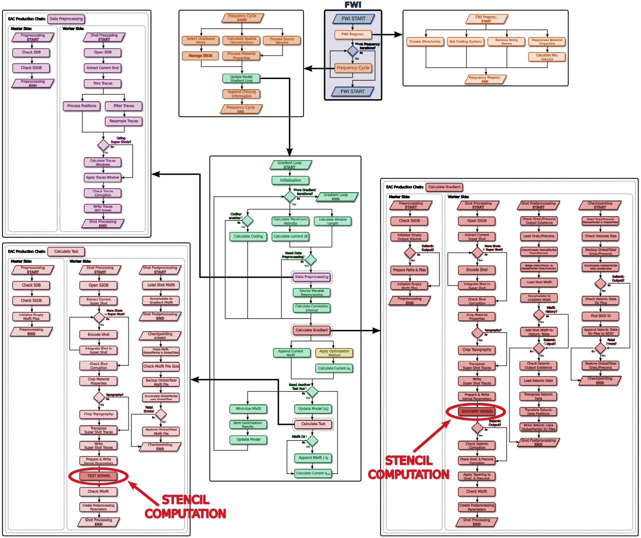
Postprocess output:
- Compute ground accumulation (cdry and cwet)
- Compute mass lost at boundaries, mass at volume and deposited
- Add divergence correction (mass-conservation law)

Write Output

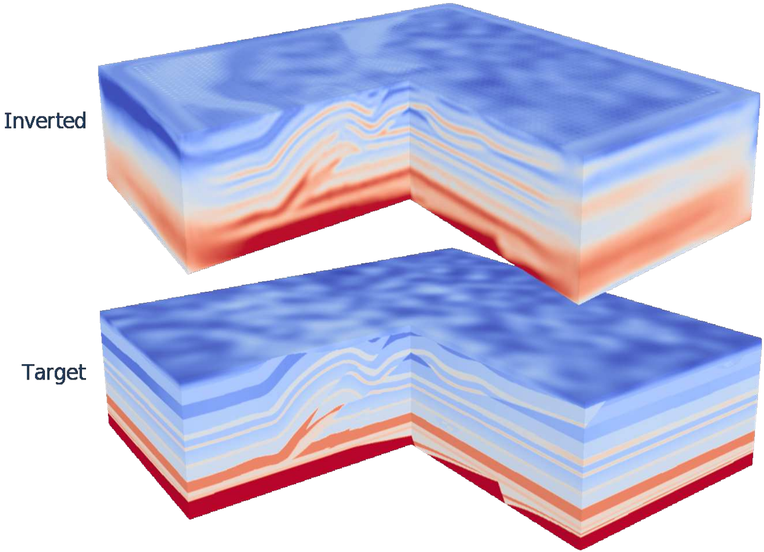
Write output:
- Scale factor, LON-LAT to Cartesian
- Write wet/dry deposit load (kg/cm2), flight level (flm), thickness deposit (cm), PM05/10/20 z-cummulative concentration (gr/cm2), 3D-volumes

Full Waveform Inversion

- Invert input signal to match a reference by modifiying the input model
- Iterative system running through different frequencies



Full Waveform Inversion - Example



Exaflop challenges

Full Waveform Inversion

- Stencil: 8th order in space, 2nd order in time, staggered grids and many params
- Resources: 5000 shots x 5 frequencies x 20 gradient iters x 5 stencil kernels x 5 hours/stencil (using 10 MN3 nodes) = 12500000 hours (200 million core/hours)
- Real synthetic case: 1601x1601x601 points $\rightarrow \approx$ 750Gb Memory per shot
- Dataset:

	Acoustic	Elastic (Velocity)	Elastic (Stress)
Material properties	1 (Vp field)	1 density (rho)	21 elastic coef.
Computational parameters	3 (p1,p2,p3 fields)	12 + 24 = 36	idem
Q (attenuation)	-	-	24 x 3 = 72
Total volumes	4	37	129

Ensemble forecasting for Ash Transport models

- Combine ADD models along with forecast models online (two-way feedback)
- Numerical prediction method generating probabilistic future states of a dynamic system
- Multiple simulations are conducted with slightly different initial conditions

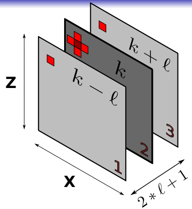
Divide and Conquer

- Alleviate footprint and register pressure by splitting internal loops (fission)
- By doing this, conflict (related with cache associativity) and capacity misses (related with cache size) are reduced
- Loop fission can be performed in Z cols, Z-X planes or in whole Z-X-Y doms
- Prefetchers can evict data which might be reused in next iterations

```
! Compute main stencil
do k = 1, NY
  do j = 1, NX
    ! Compute Advection term (no slope-lim)
    ! Single loop performing everything
    do i = 1, NZ
      c(i,j,k,0) = c(i,j,k,1) -
        (dtddz * (u(i+1,j,k) * c(i+1,j,k) -
                  u(i-1,j,k) * c(i-1,j,k))
         dtddx * (v(i,j+1,k) * c(i,j+1,k) -
                  v(i,j-1,k) * c(i,j+1,k))
         dtddy * (w(i,j,k+1) * c(i,j,k+1) -
                  w(i,j,k-1) * c(i,j,k-1))
    end do
  end do
end do
```

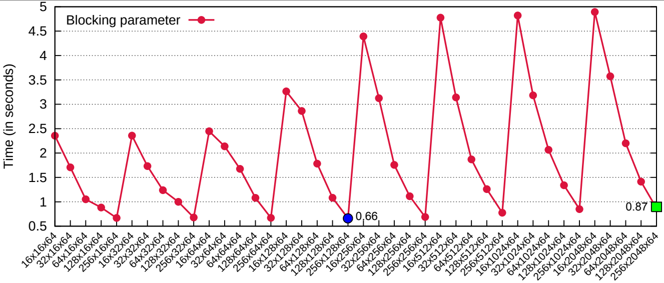
```
! Compute main stencil
do k = 1, NY
  do j = 1, NX
    do i = 1, NZ  ! First block
      c(i,j,k,0) = c(i,j,k,1) -
        (dtddz * (u(i+1,j,k) * c(i+1,j,k) -
                  u(i-1,j,k) * c(i-1,j,k)))
    end do
    do i = 1, NZ  ! Second block
      c(i,j,k,0) = c(i,j,k,0) -
        (dtddx * (v(i,j+1,k) * c(i,j+1,k) -
                  v(i,j-1,k) * c(i,j+1,k))
         dtddy * (w(i,j,k+1) * c(i,j,k+1) -
                  w(i,j,k-1) * c(i,j,k-1)))
    end do
  end do
end do
```

- Time-blocking algorithms pose many issues in Industrial codes (execution flow very complex between time-steps)
- The first two dimensions (Z and X) are what really matters
- Space-blocking (a.k.a. Rivera) shows partial effectiveness and clearly depends on Cache and Z-X dimensions per thread



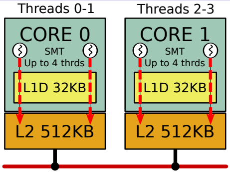
Ash Dispersion model: MareNostrum 3. 32KB L1 D-Cache, 256KB L2 Cache and 20MB L3 Cache

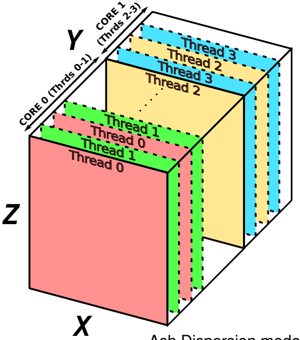
Domain size	256x256x64	256x512x64	256x1024x64	256x2048x64
No blocking	0.077	0.158	0.428	0.873412
Best blocking	0.075 (256x128)	0.151 (256x128)	0.334 (256x128)	0.66 (256x64)
Speedup	1.02x	1.05x	1.27x	**1.32x**



Thread & Domain Affinity - SMT Approach

- SMT enables sharing cache resources among threads
- D-Cache reuse may be improved by a wise decomposition
- Reduce conflict & capacity misses due to smaller footprint






```
# pragma omp parallel firstprivate(iniy,endy)
{
  int tid = omp_get_thread_num()%THRCORE;
  int dom = omp_get_max_threads()/THRCORE;
  int nby = (endy - iniy) / dom;
  int rnb = (endy - iniy) % dom;
  if (rnb > dom) iniy = iniy + (++nby)*dom;
  else           iniy = iniy + nby*dom + rnb;
  endy = MIN(iniy+nby, endy);

  /* Update maingrid */
  for (k= iniy+tid; k< endy; k+= THRCORE) {
    for (j= inix; j< endx; j++) {
      for (i= iniz; i< endz; i++) {
```

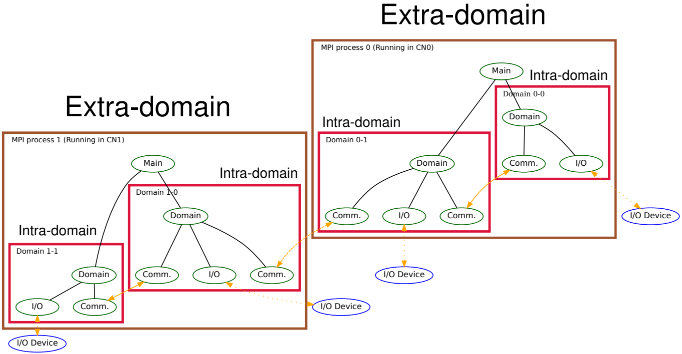
Ash Dispersion model: Intel MIC, 4 cores (2 threads each). Domain 256x64x1024

Metric	L1 Misses	L2 Demand + L2 HWP Misses	TLB Misses	Time (in sec)
No affinity	9504992	6245341 *(3492094 + 2753247)*	4411206	4.88
With affinity	8201066	5294566 *(361439 + 4933127)*	346578	3.96
Reduction	14%	16%	92%	**19%**

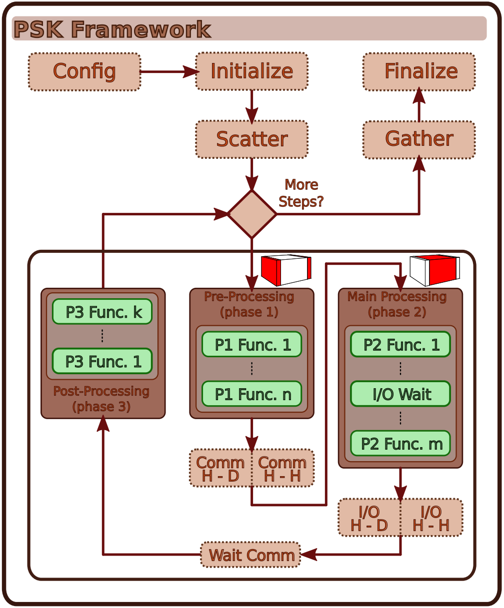
WARIS

- Multi-purpose framework to solve FD problems
- Developed in a parallel and efficient way for ES and CFD problems
- Structured meshes (conforming and non-conforming)
- Modular to ease development cycles, portability and reusability
- Supports Explicit, Implicit and Semi-implicit methods

- High level of parallelism
- Asynchronous I/O
- Computation & I/O overlapping



WARIS - PSK Framework

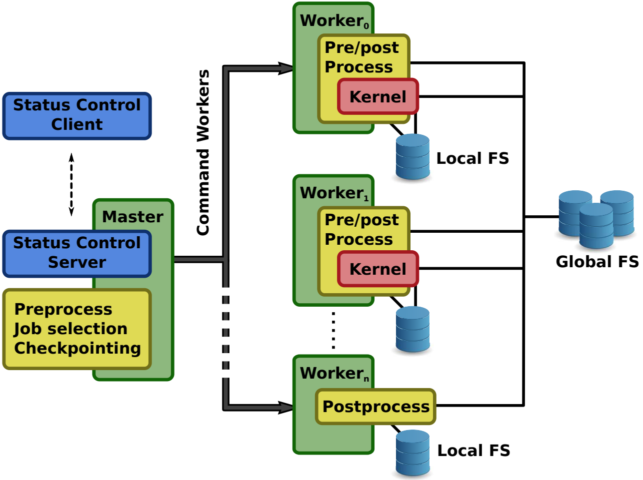


- Generalized framework providing an execution flow
- User must create problem specialization routines
- Red blocks are provided by the PSK Framework
- Green blocks are provided by the user problem specialization
- P1 routines compute data required for domain decomposition exchange
- P2 routines perform computation over the remaining domain

Embarrassingly parallel executions

- Process several physical problems in parallel
- Statistical study, optimal parameter search or production chain

- Resilience
- Fault tolerance
- Postprocessing
- Checkpointing



Example of success: WARIS-Transport vs FALL3D

WARIS-Transport explicit kernel optimizations

- SIMDization (SSE, AVX and MIC)
- Auto-tune of blocking parameter
- Pipeline optimizations (loop fission)
- Parallel I/O (active buffering strategy & two-phase collective)

Eyjafjallajökull validation testcase

- Input dataset: 9 GBytes of meteorological data
- Output dataset: 200 MBytes of simulated concentration data
- FALL3D requires 1h and 58 minutes with 16 processors
- WARIS-Transport took 17 minutes to process it (Speedup 6.8×)

Eyjafjallajökull case. Domain 41*x*241*x*141. Test conducted in MareNostrum (Intel SandyBridge-EP E5-2670)

Number	Explicit Kernel		Meteo data		Output		Total	Speed-up
Proc.	P1 stage	P2 stage	I/O	Postprocess	Preprocess	I/O	Time	
1	0.0	4028.1	153.4	2599.5	11.0	3.9	8654.5	1.0
2	54.6	1972.9	103.6	1335.2	4.8	2.8	4350.0	1.9
4	55.1	971.1	83.9	718.9	2.5	2.9	2318.5	3.7
8	57.5	448.3	62.4	414.9	1.2	0.2	1304.9	6.6
16	67.4	296.8	90.2	257.2	0.7	0.3	1037.8	8.3
24	68.9	121.61	92.2	201.2	0.4	9.5	788.0	10.9

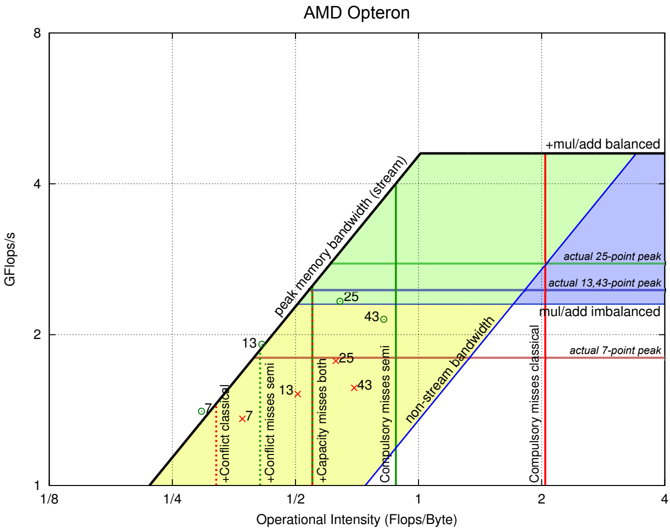
Summary & Conclusions

- Optimize the stencil computation is important, BUT
- Not all optimization techniques in academia give a leap
- Space-tiling is only useful for certain dataset cases and caches
- Consider underlying hardware when DD (SMT & shared caches)
- Design a proper framework for Industrial codes
- Include Auto-tuning techniques to your framework (Model + Search parameter)
- Keep a simple domain decomposition to reduce memory copies
- Independent threads for independent tasks
- Overlap Communication - Computation - I/O tasks
- Include a production chain for embarrasingly parallel cases

Questions


Thank you for your attention!
Copyright 2013. Barcelona Supercomputing Center - BSC

Modelling & Hardware Limitations



Permute or not permute, that's the question

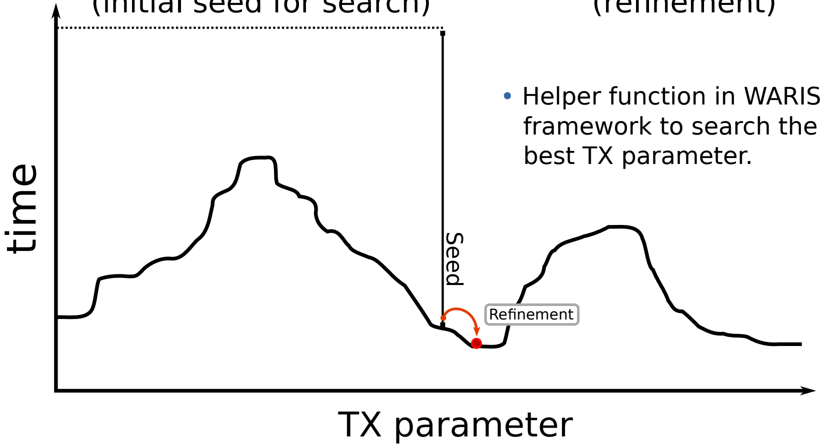
Performance may vary depending on the first two axis sizes (Z and X)

- Is it 256x2048x64 traversing computation $\Leftrightarrow$ 256x64x2048?
- Permute input and output buffers
- Does the permutation pay off the cost and benefit?

Auto-tuning blocking parameters

Prediction model
(initial seed for search)

Modeling +
Fine Grain Search
(refinement)



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