High Level Transforms for SIMD and low-level computer vision algorithms

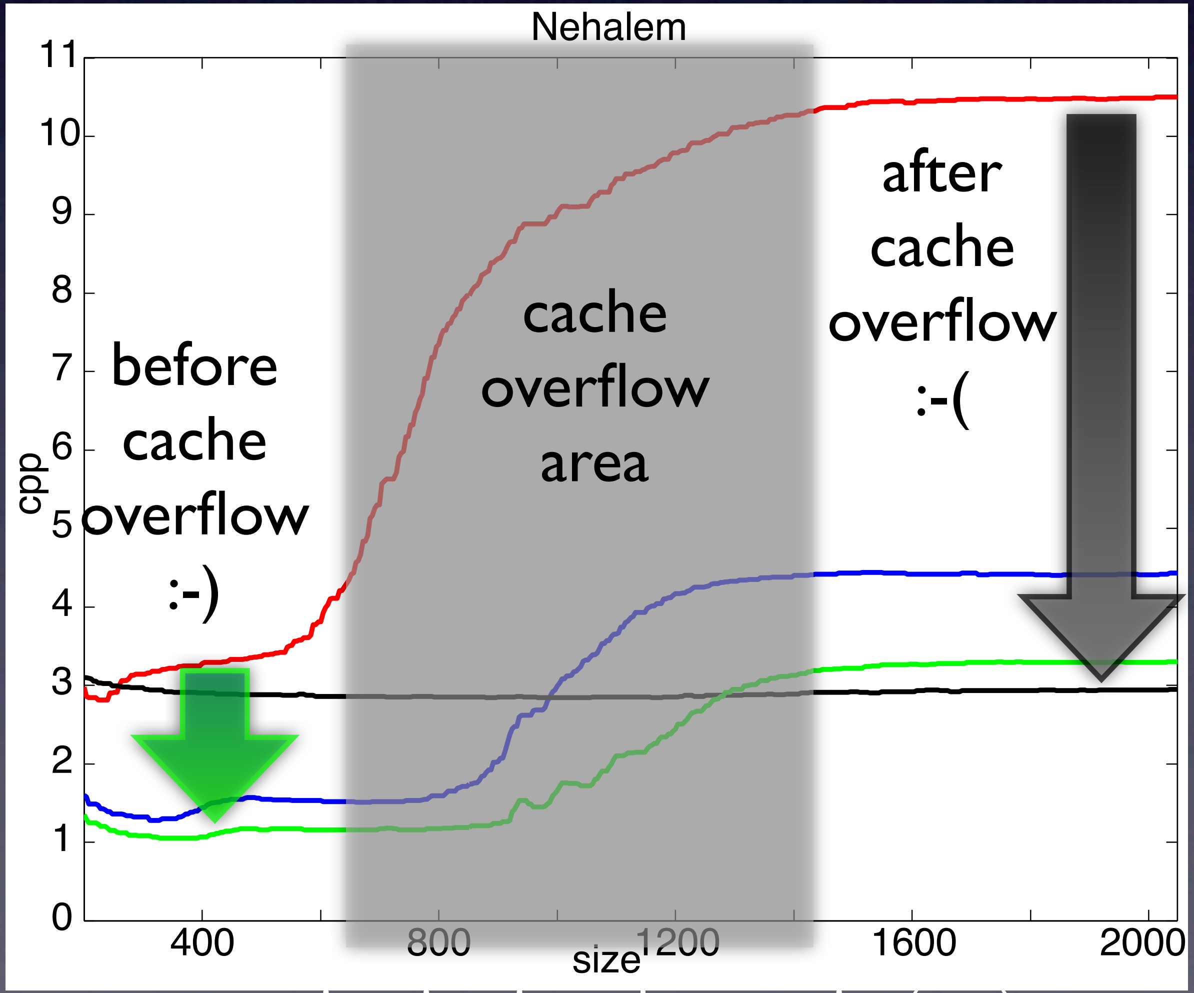
L. Lacassagne, D. Etiemble, A. Zaharee,
A. Dominguez, P. Vezolle

extended version of [PPoPP/WMVP 2014]

lionel.lacassagne@lip6.fr
www.lip6.fr

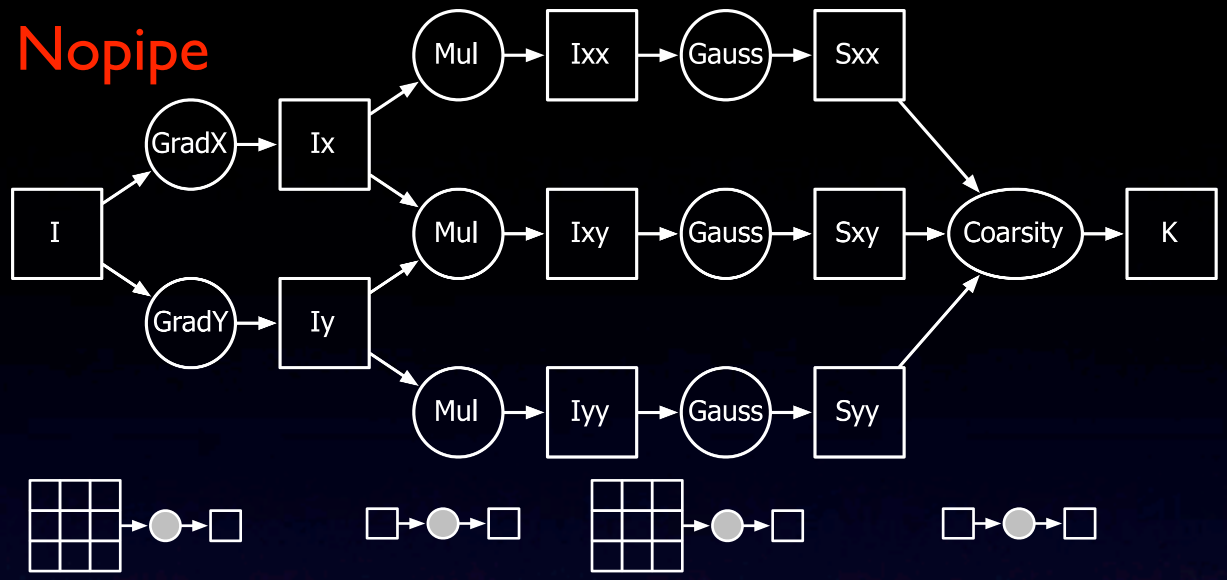
Context & HLT

- Context:
  - implementation of 2D stencils and convolution for image processing
- Observation : cache overflow
  - SIMD+OpenMP is not enough to get and sustain a high level of performance (here cycle per point vs image size)
  - after cache overflow (capacity problem), the perf is divided by 3 (at least!)
- HLT (High Level Transforms)
  - to get better performance before CO
  - to sustain performance after CO (Intel mobile proc, ARM Cortex A9)
- Presentation in 3 points
  - algorithm presentation
  - algorithm optimization with HLT
  - Benchmarks of Intel, IBM and ARM machines



execution time in cycle per point (cpp)
the lower the better

Harris Point of Interest detector

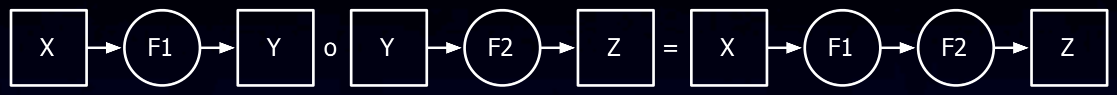


- Harris is representative of low-level image processing
  - combination of point and convolution operators = 2D stencils
    can scale across a wide range of codes using 2D stencils and convolutions
  - arithmetic intensity is low => memory bound algorithm like for many image processing algorithms
  - neither SIMD nor OpenMP will change that !
  - Apply High Level Transforms to get a higher level of performance

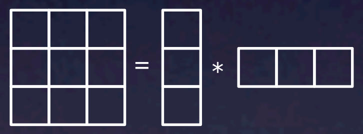
operator	MUL+ADD	LOAD+STORE	arithmetic intensity AI
Nopipe	22+35=57	54+9=63	0.90

HLT = algorithmic transforms

- First set of HLT:
  - Operator Fusion (named Halfpipe and Fullpipe)
    - to avoir memory access to temporary array (less stress on memory buses)

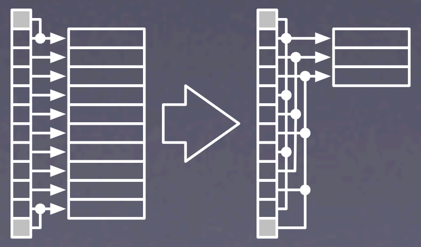


  - Convolution decomposition with column-wise reduction (named *Red*)
    - to both reduce arithmetic complexity and memory access amount



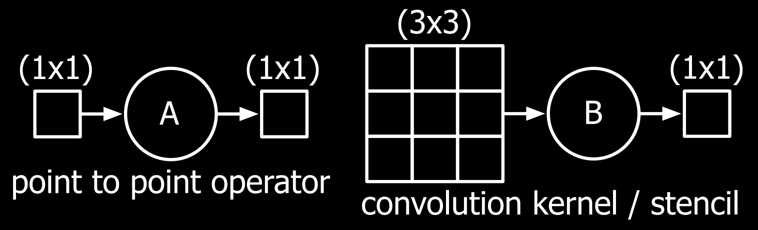
  - Can be done in {scalar, SIMD} x {mono-thread, multi-threaded}

- Second set of HLT
  - Operator pipeline + circular buffer (to store temporary data) with Modular addressing (Mod)
    - increases spatial & temporal locality
  - but memory layout must be transformed



operator fusion: transform rules

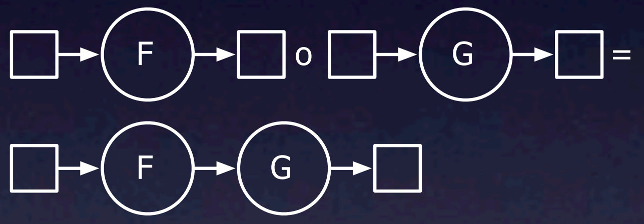
1. operators are described with producer-consummer model with input pattern and output pattern



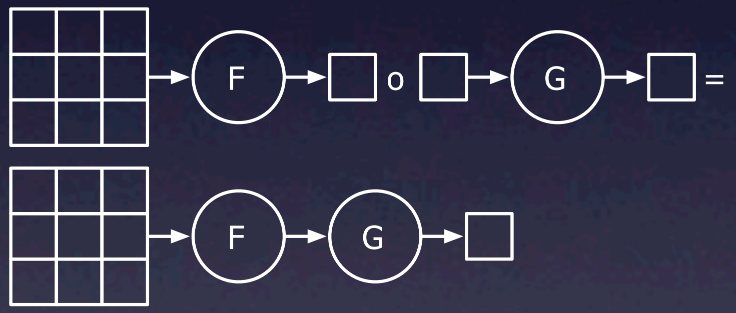
2. pattern adaptation: input of 2nd operator = output of 1st operator can be combined with pattern transformations: factorization / decomposition / combination

3. scalarization: temporary results are stored into registers (variables instead of array cells)

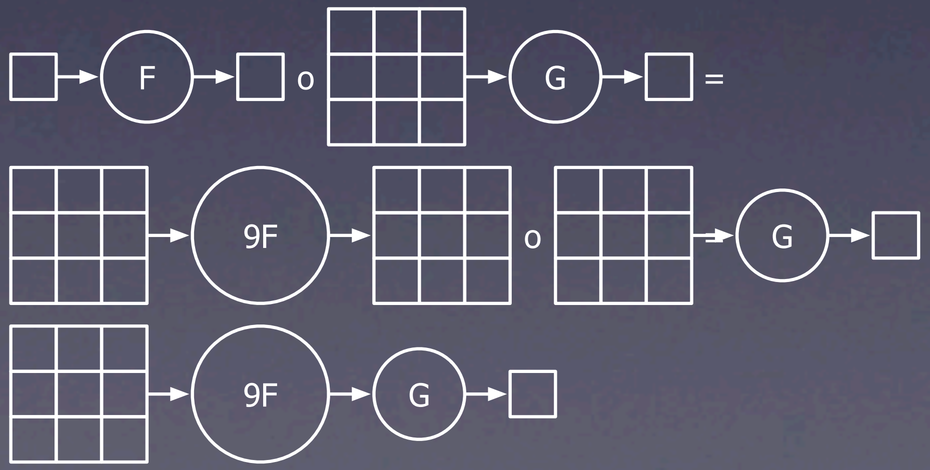
case #1 (1x1)->(1x1) o (1x1)->(1x1) = (1x1)->(1x1)



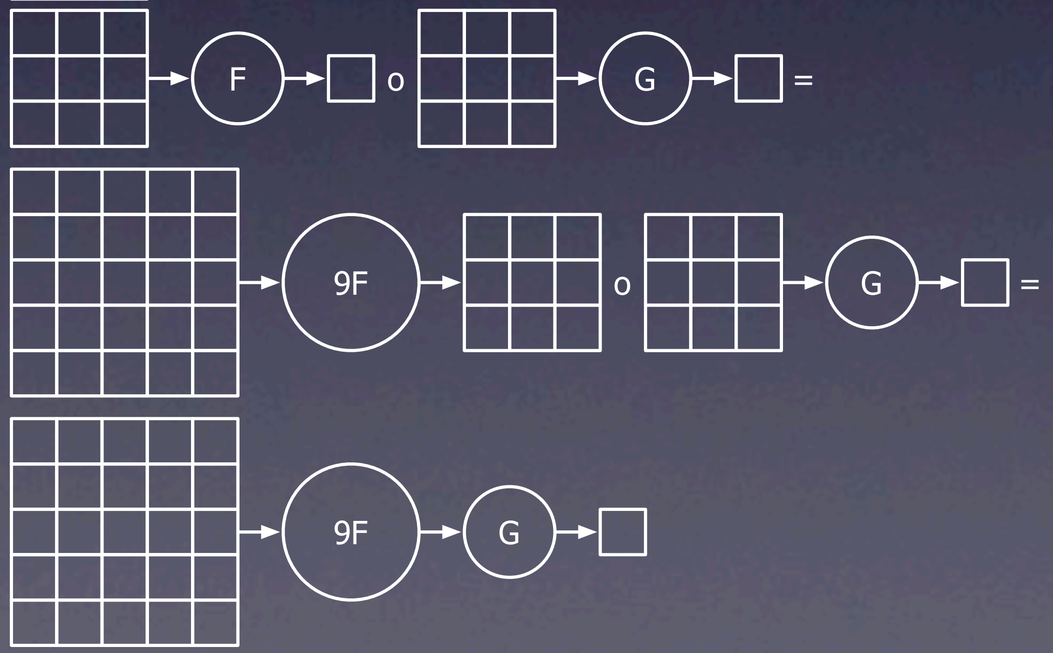
case #2 (3x3)->(1x1) o (1x1)->(1x1) = (3x3)->(1x1)



case #3 (1x1)->(1x1) o (3x3)->(1x1) = (3x3)->(1x1)

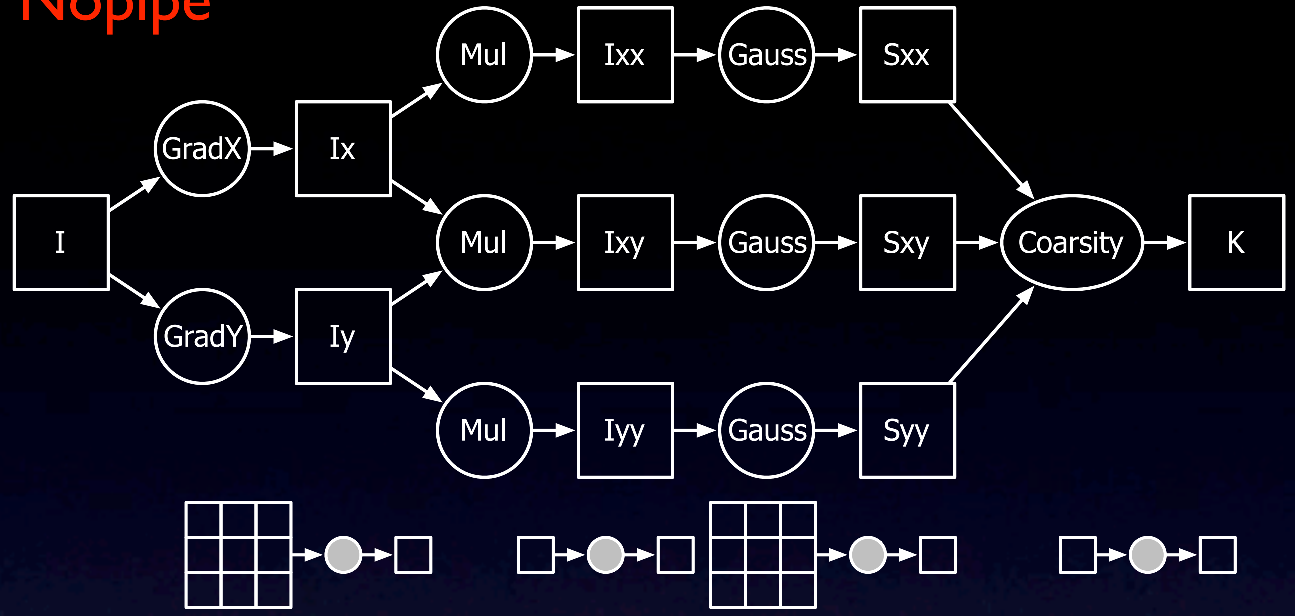


case #4 (3x3)->(1x1) o (3x3)->(1x1) = (5x5)->(1x1)

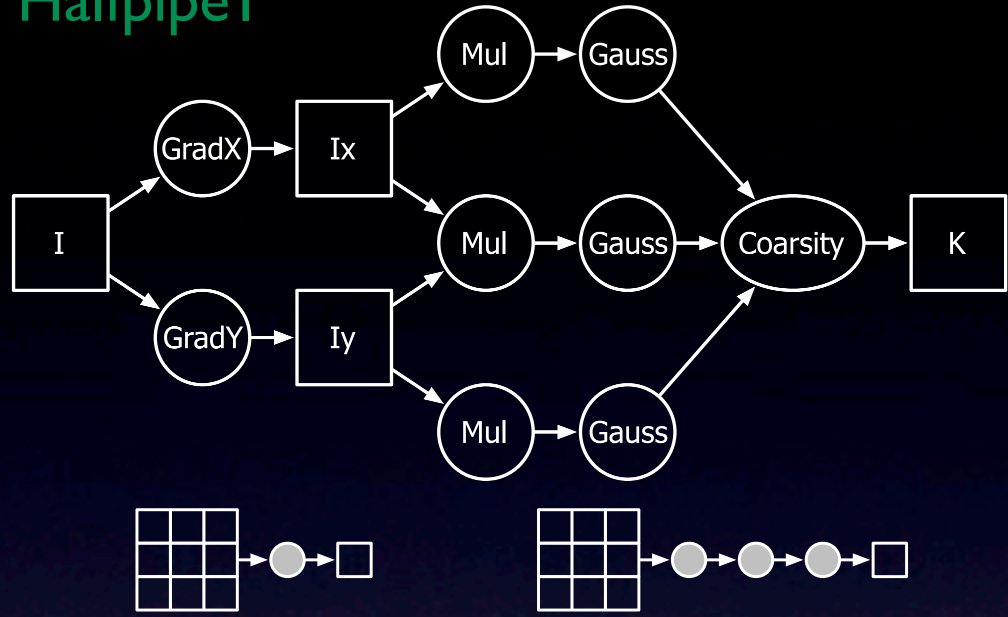


Harris & operator fusion

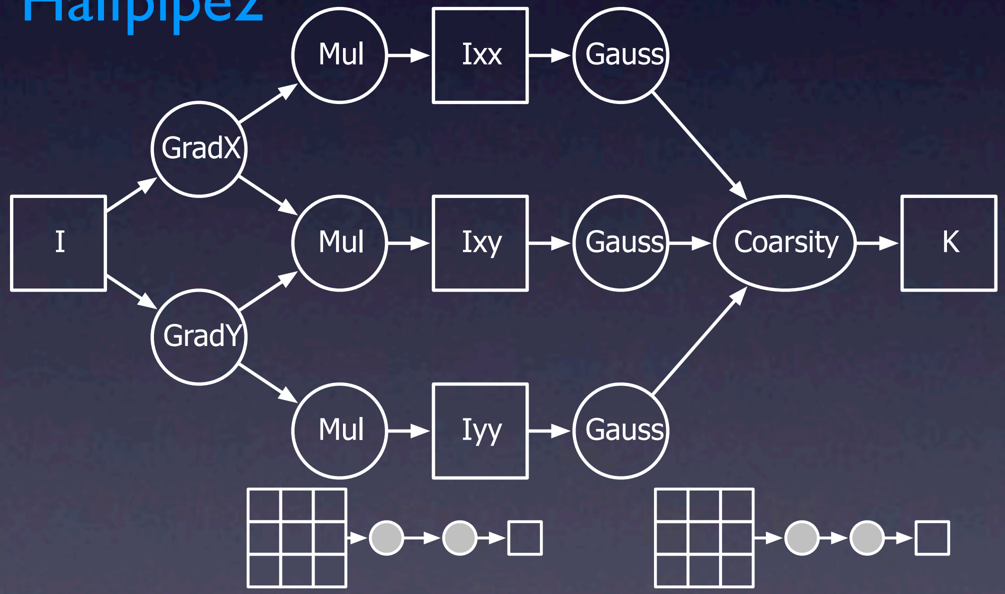
Nopipe



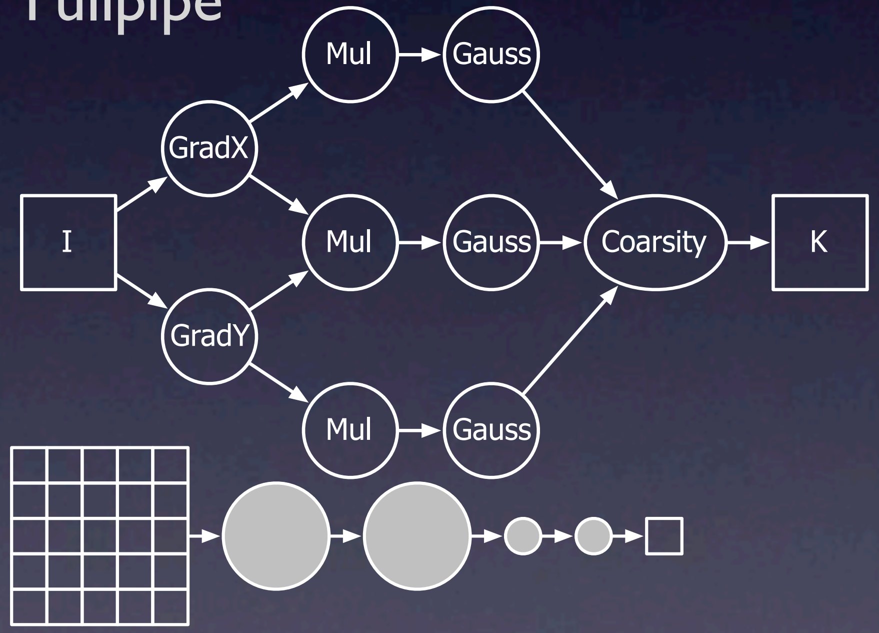
Halfpipe1



Halfpipe2



Fullpipe



operator	MUL+ADD	LOAD+STORE	AI
Nopipe+red	5+27=32	21+9=30	1.10
Halfpipe2+red	5+27=32	12+4=16	2.0
Halfpipe1+red	11+27=73	27+3=30	2.4
Fullpipe+red	29+82=111	5+1=6	18.5

⇦ memory bound ?

⇦ computation bound ?

Macro Meta Programming

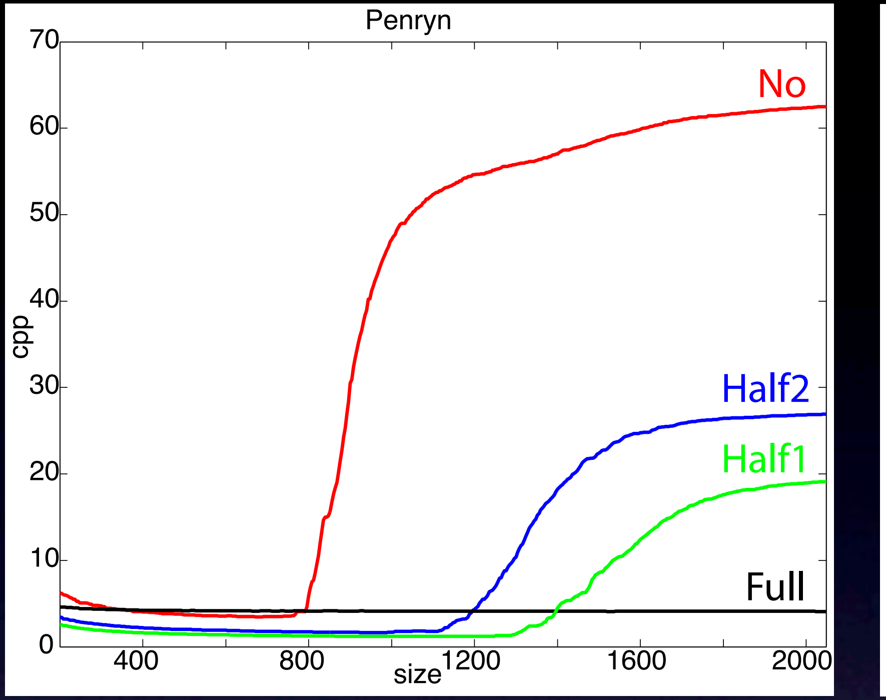
- Level #1
  - one set of macros for each SIMD instruction set
    - IBM Altivec (aka VMX), SPX
    - Intel SSE, AVX, AVX2, KNC
    - ARM Neon
    - ST Microelectronics VECx
  - Arithmetic: vec_add vec_sub, vec_mul, vec_fmadd, vec_fmsub, vec_set
    I/O: vec_load1D, vec_store1D, vec_load2D, vec_store2D
    permutation: vec_left1, vec_right1
- Level #2
  - one set of Harris operators: GRADIENTX, GRADIENTY, MUL, GAUSS, COARSITY

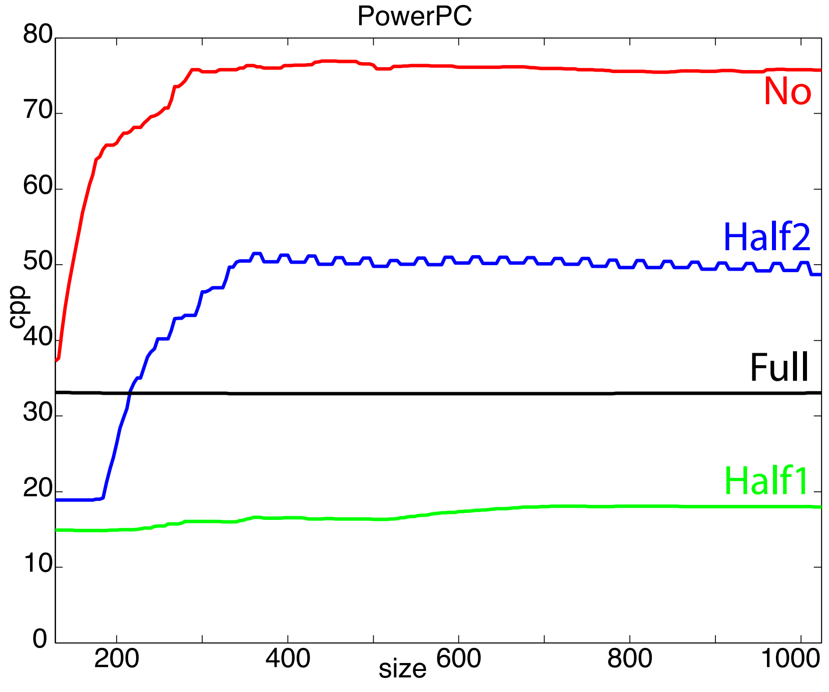
Three families of processors

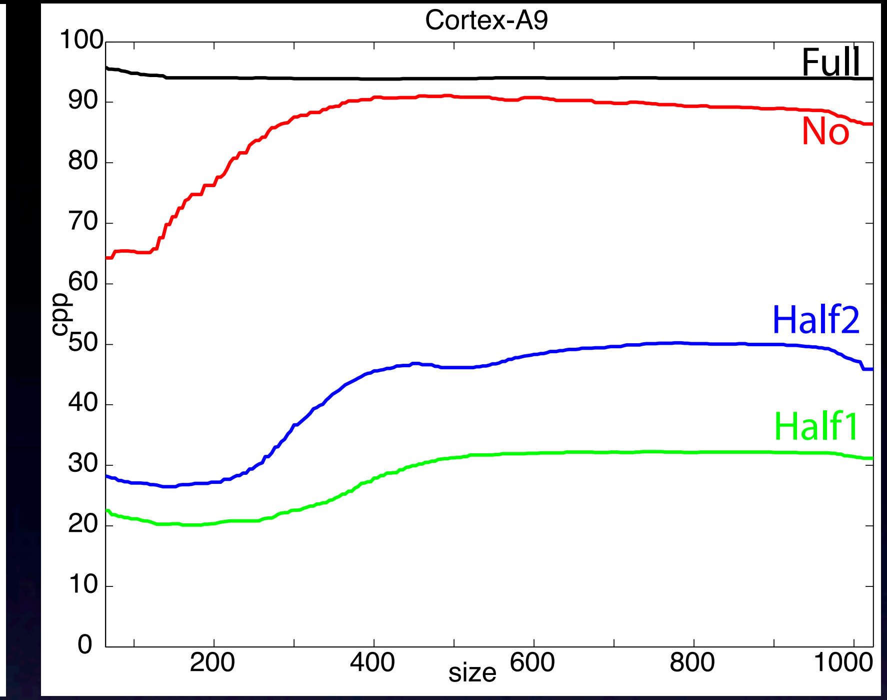
- Three processor famillies: Intel, IBM and ARM
  - Intel (Core and Xeon): Penryn, Nehalem, SandyBridge, IvyBridge, Haswell + preliminary result on Xeon-Phi
  - IBM: PowerPC 970MP, Power6, Power7, Power7+
  - ARM: Cortex-A9 (TI OMAP4), Cortex-A15 (Samsung Exynos 5)
  - Prediction: depending on processor's Arithmetic Intensity, cache overflow magnitude will be high or low (same for the impact of HLT)
- SIMD & fairness: 128-bit only to be fair with ARM
  - paper is about HLT impact, not architecture comparison (except SIMD multi-cores vs GPU)
  - SIMD = {SSE, Altivec, Neon} without FMA, compiler = {icc, xlc, armgcc}
  - All codes are fully parallelized with SIMD & OpenMP
  - Xeon-Phi codes use 512-bit MIC instructions

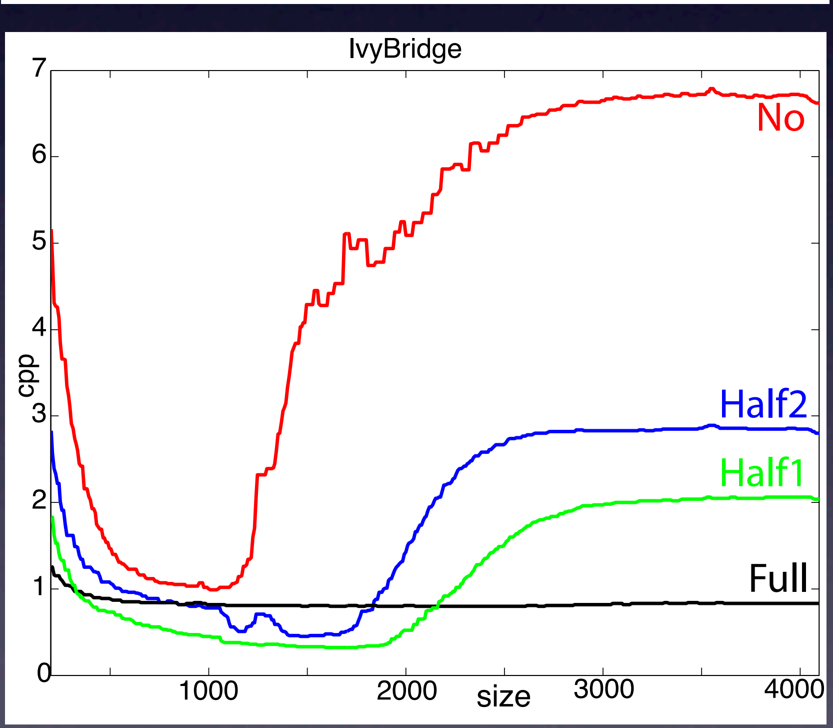
proc	# cores	GHz	GFlops	BW (GB/s)	AI
Cortex A9	1x2	1.2	4.8	1.2	4.0
Cortex A15	1x2	1.7	13.6	5.8	2.3
PowerPC	2x2	2.5	40	5.4	7.4
Power7+	4x8	3.8	486	265	1.8
Nehalem	2x4	2.67	85.1	22	3.9
IvyBridge	2x12	2.7	518.4	92	5.6
Xeon Phi	1x61	1.33	1298	170	7.6

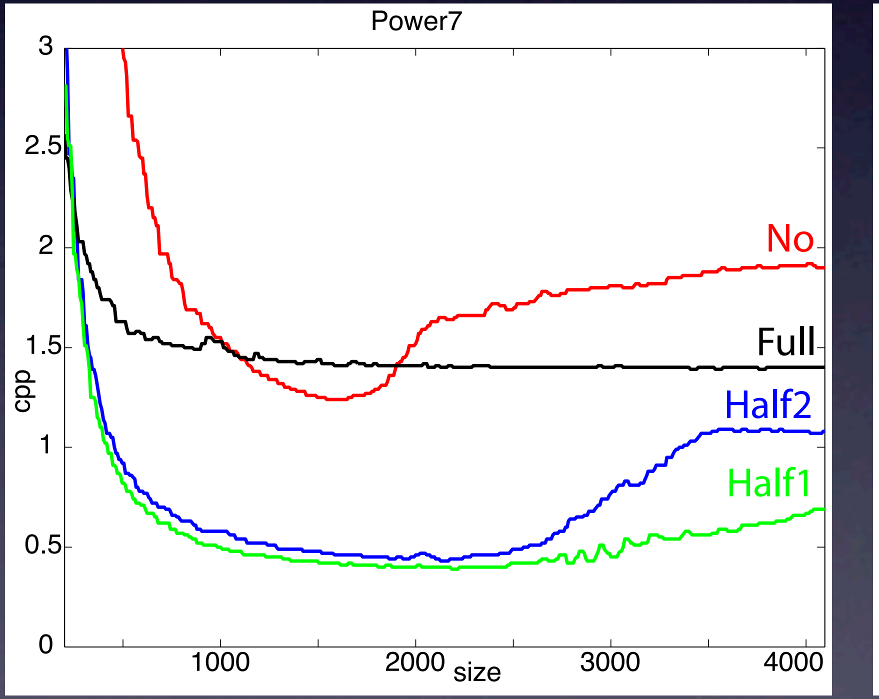
Benchmark: Halfpipe & Fullpipe with reduction

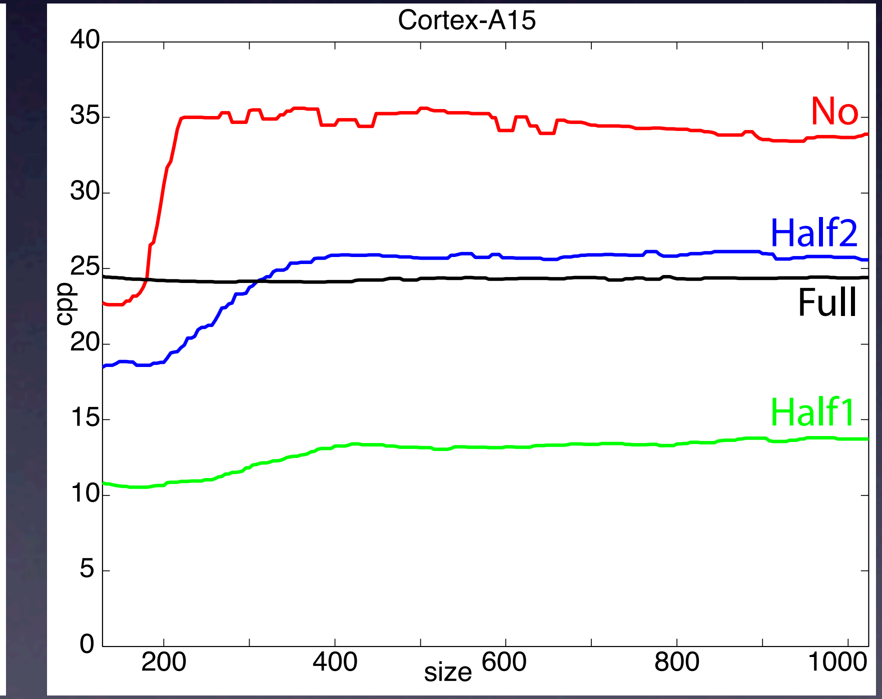








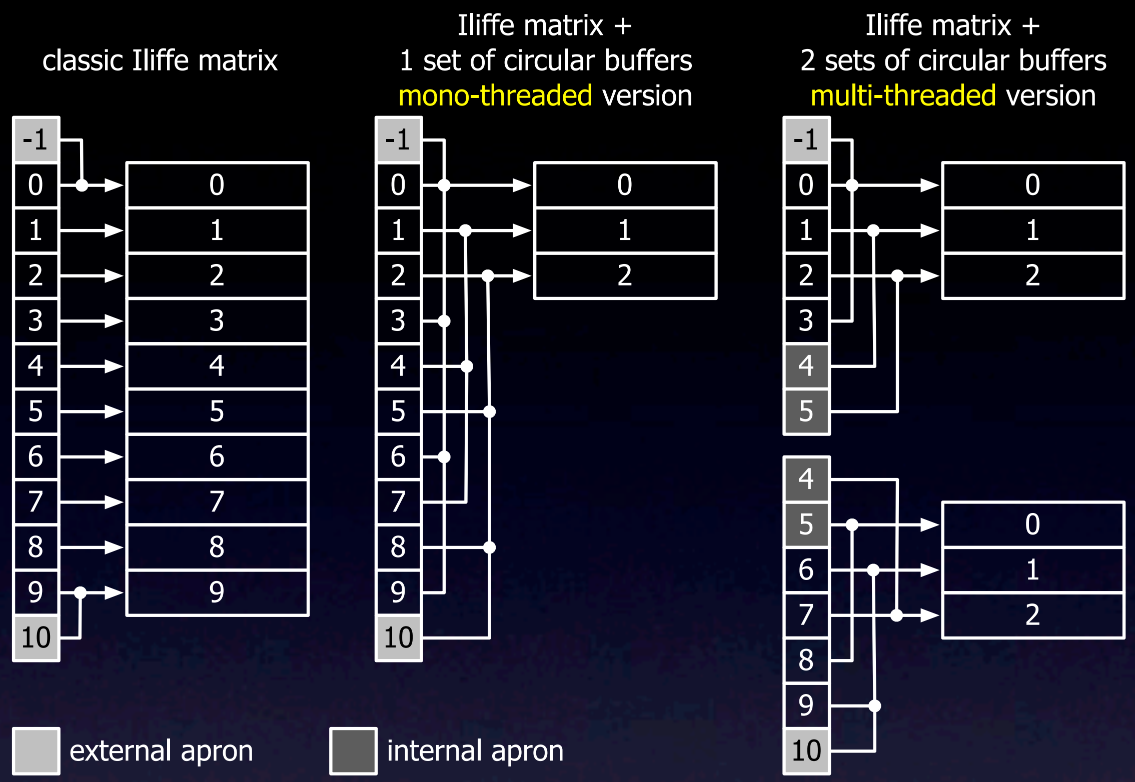




- metric: cpp = cycle per pixel vs matrix size [32x32 .. 4096x4096]
- For all: data fit in cache longer, Halpipe1 is always faster than Halfpipe2
- depending on processor's AI, Fullpipe is faster that Halfpipe1
- Halfpipe2 and Halfpipe1 are memory bound => cache overflow still happens :-(

Modular addressing #1

- Illife matrix [1961]:
  - Popularized by Numerical Recipes in C
  - offset addressing = arrays of pointers to rows
  - can easily manage apron for stencil / convolution
  - hypothesis:
    - p parallelism: p cores, p threads
    - k=3 for (k x k) stencils / convolutions

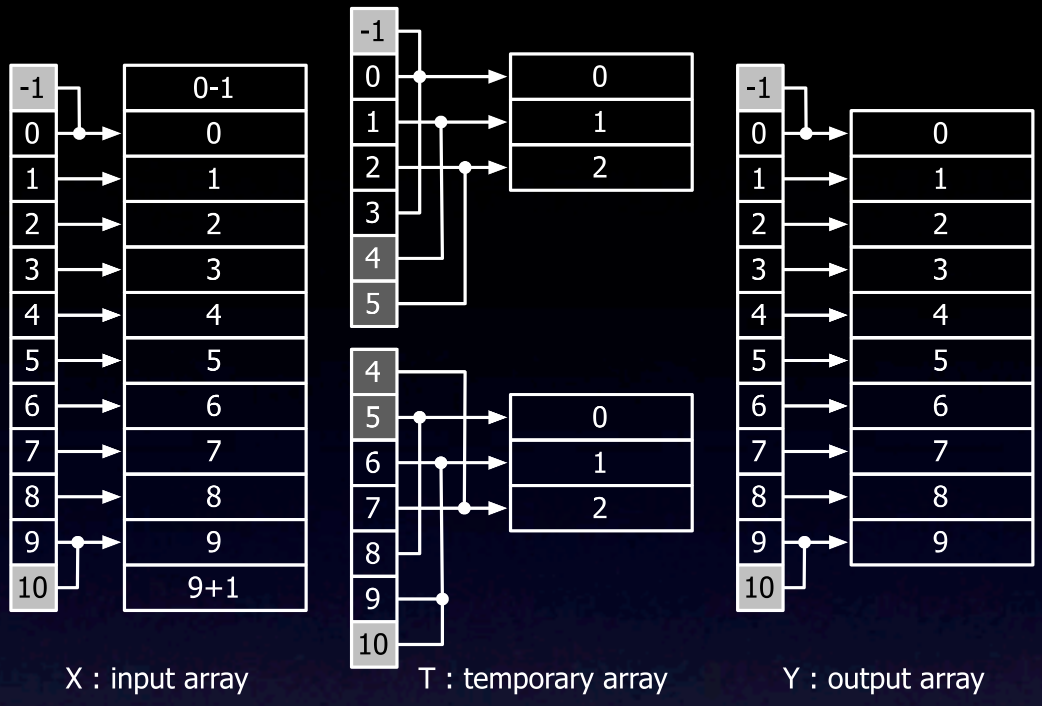


- Circular buffer with modular addressing
  - mono-thread: 1 set of 3-row circular buffer for (3x3) stencil: T[i] points to i mod 3
  - multi-thread version: p sets of p set of 3-rows circular buffers:
    for "internal apron" T[i] points to different sets of circular buffers

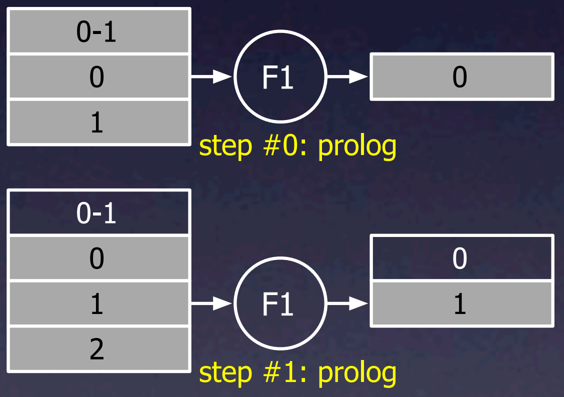
- Hack
  - the memory layout is transparent to the user ... even for multi-threaded version :-)
    (thanks to offset addressing)
  - on only has to write a pipelined version of the operators

Modular addressing #2

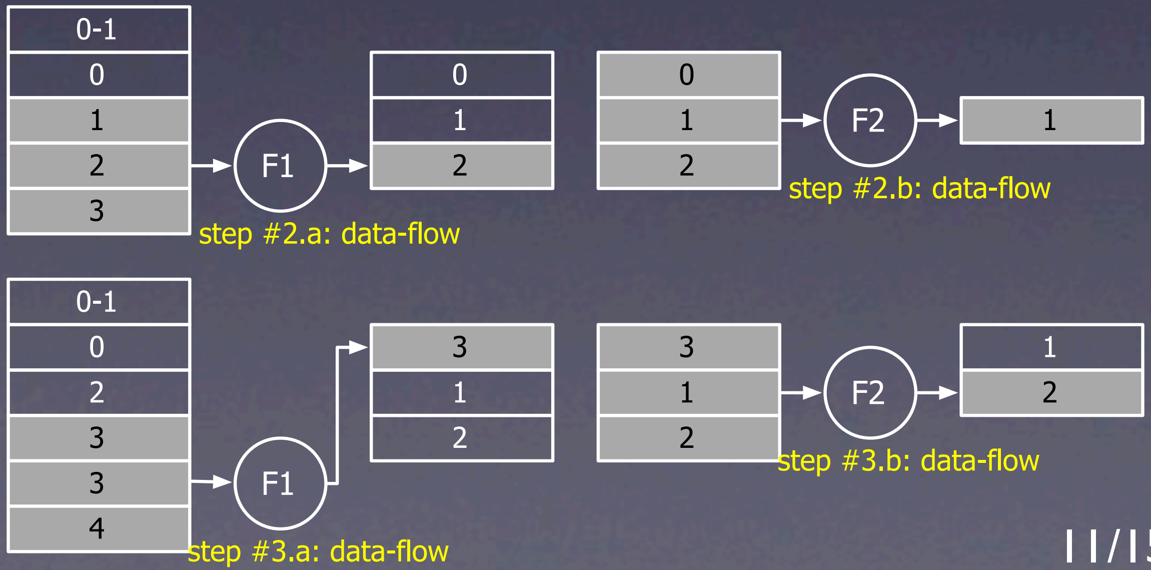
- Example:
  - input and output arrays X, Y should be classical arrays
  - Temporary array T is a set(s) of circular buffer(s)
  - pipeline of two convolution operators: F1 and F2
  - apron processing is not detailed in order to simplify



- Prolog: initiate the filling of T
  - call operator F1 (k-1) times: steps #0, #1
    to produce 2 rows

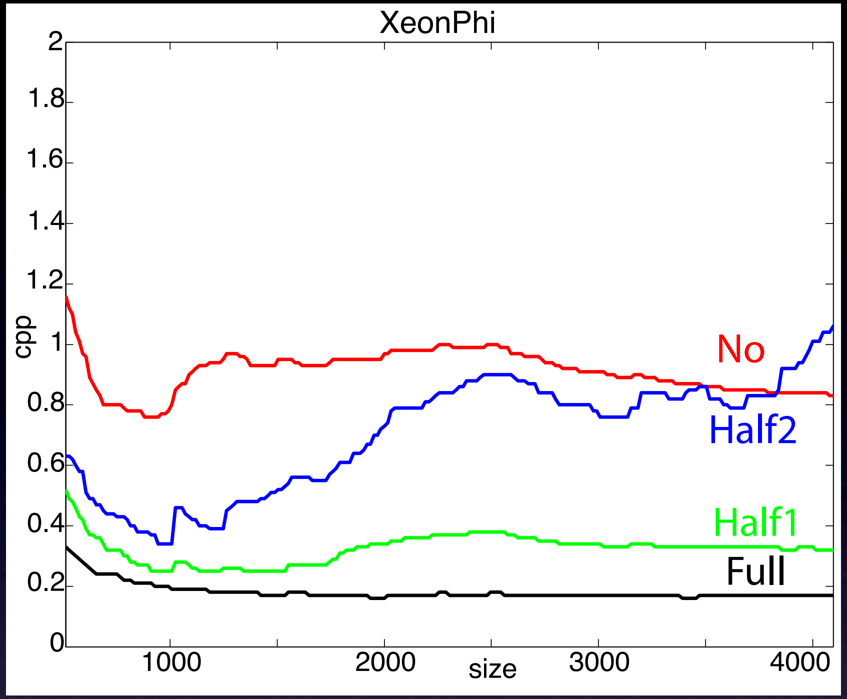


- Data-flow processing: for each input, 1output
  - after each call of F1, there is enough data to call F2
  - repeat until the end (size / p)



Xeon Phi: preliminary results

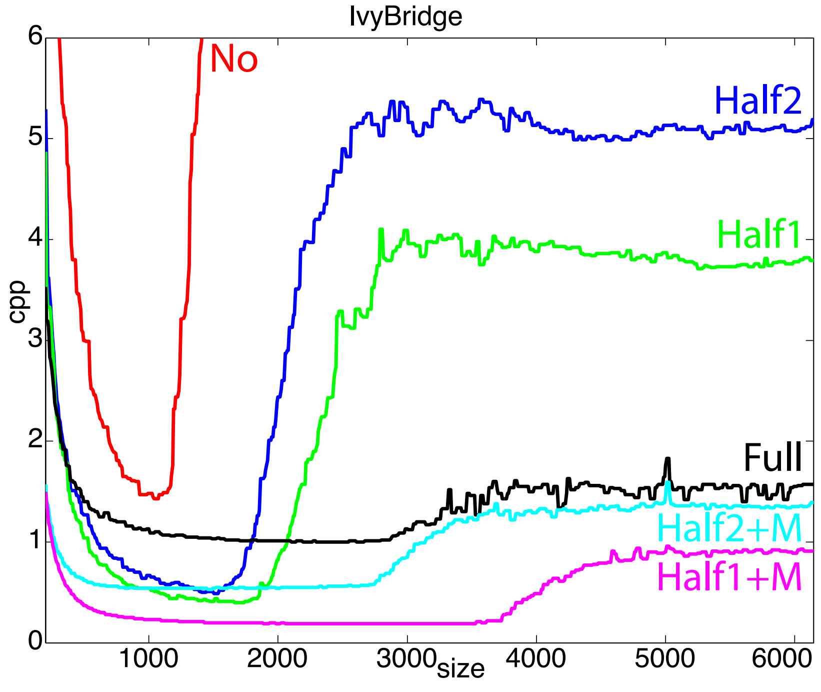
- Observations
  - Mod results not presented: some problems with memory and multi-threading to fix (with Vtune) ...
  - programing model is easy (shared memory + SIMD)
  - HLT transforms are efficient: x5.5
  - peak bandwidth is reached

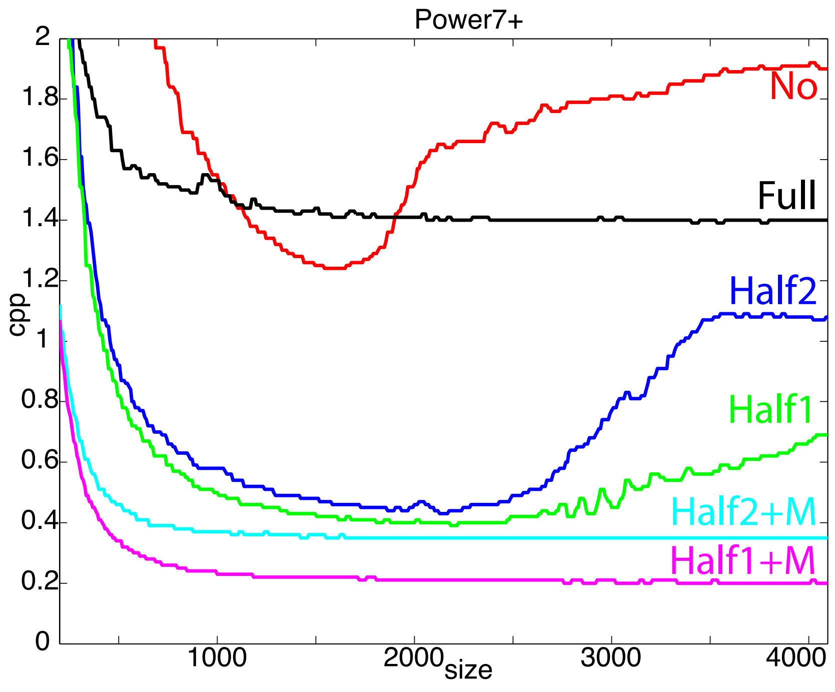


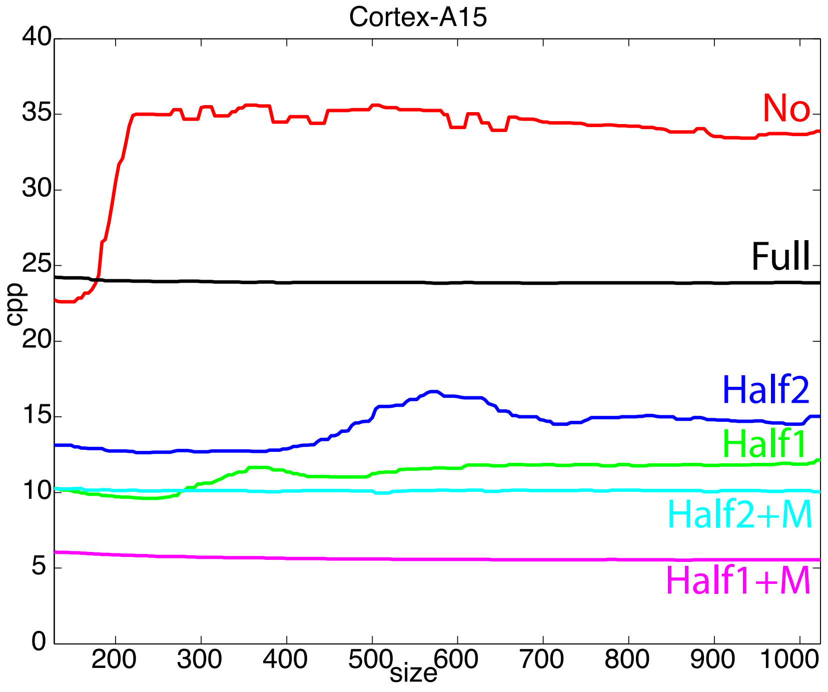
proc	Nopipe	Half2+red	Half1+Red	Full+red	speedup
cpp	0.99	0.81	0.35	0.18	x5.5
GFlops	65.8	52.5	144.4	820.2	
BW (GB/s)	306	105	182	177	

Xeon-Phi: cpp, GFLops and Bandwidth for 2048x2048 images

Benchmark: Modular addressing







	cpp			HLT speedups		
proc	no	red	mod	red	mod	tot
Cortex A9	86.4	31.2	9.4	x2.8	x3.3	x9.2
Cortex A15	34.1	13.9	5.6	x2.5	x2.5	x6.1
PowerPC	75.7	18.0	10.2	x4.2	x1.8	x7.4
Power7+	1.62	0.40	0.21	x4.1	x1.9	x7.7
Nehalem	10.5	2.95	0.50	x3.6	x5.9	x21.0
IvyBridge	5.30	0.65	0.15	x8.2	x4.3	x35.3
Xeon Phi	0.99	0.18	-	x5.5	-	-

- Mod codes sustain the performance longer after initial cache overflow.
- On IvyBridge, even for Halfpipe1+Red+Mod, cache overflow still happens ...
- The total speedups are very high: x9.2 & x7.7 for A9 & P7 up to x35.3 for IVB (depends on proc AI)
- Fullpipe rank is a clue to processor peak power (the lower, the higher)

Multi-core SIMD versus GPU

- GPU
  - benchmark done on GTX 580 and estimated for Titan and K40 for 2048x2048 images
  - Texture versions use free bi-linear interpolation to reduce computations
    tile size has been optimized (exhaustive search) for Shared memory
  - HLT are also efficient for GPU: x5.5
- Observations:
  - GPP match GPU performance thanks to HLT
  - for Harris, Xeon-Pi behaves like a GPU: minimize communications => Fullpipe is the fastest version

	No	Half+Red	Full+Red	Half+Red+Mod	gain
Cortex A15	84.1	34.2	60.3	13.7	x6.1
IvyBridge	8.23	1.01	1.26	0.23	x35.3
Power7+	1.79	0.44	1.5	0.23	x7.7
Xeon Phi	3.12	1.10	0.57	-	x5.5
	No	Half	Full	Full	
memory	global	Tex	Tex	Shared	
GTX 580	6.52	2.24	1.4	1.16	x5.6
Titan (est.)	*2.29*	*0.79*	*0.49*	*0.41*	*x5.6*
K40 (est.)	*2.41*	*0.83*	*0.52*	*0.43*	*x5.6*

Conclusion & future works

- Conclusion
  - huge impact of High Level Transforms for SIMD multicore GPP, GPU and Xeon-Phi
  - can scale across a wide range of codes using 2D stencils and convolutions
  - done by hand as compilers can't vectorize Red versions
  - GPP match GPU with HLT
- Future works
  - improve Xeon-Phi performance for Mod versions
  - benchmark up-coming machines (Xeon Haswell, Power8, Cortex A57, ...)
  - apply HLT to complex algorithms (image stabilization, tracking)
  - Harris code as a reference for benchmarking ?

We are looking for access to new machines (through NDA ?) and collaboration

Thanks !