Exascale Computing for Radio Astronomy: GPU or FPGA?

Kees van Berkel

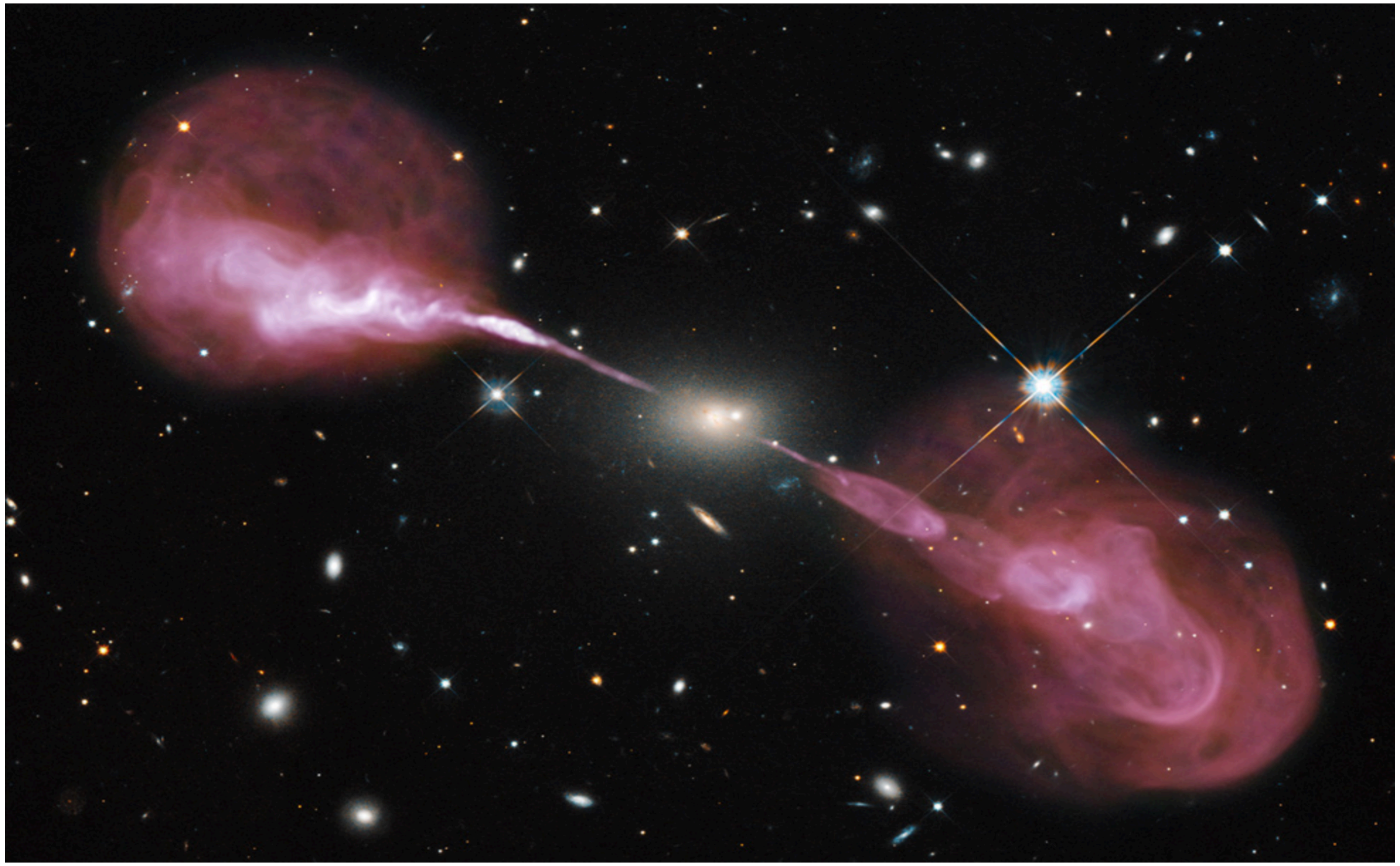
MPSoC 2016, Nara, Japan

2016, July 14

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

Radio Astronomy: Herculus A   (a.k.a. 3C 348)



"… optically invisible jets, one-and-a-half million light-years wide, dwarf the visible galaxy from which they emerge."

Image courtesy of NRAO/AUI

VLA radio telescope, New Mexico

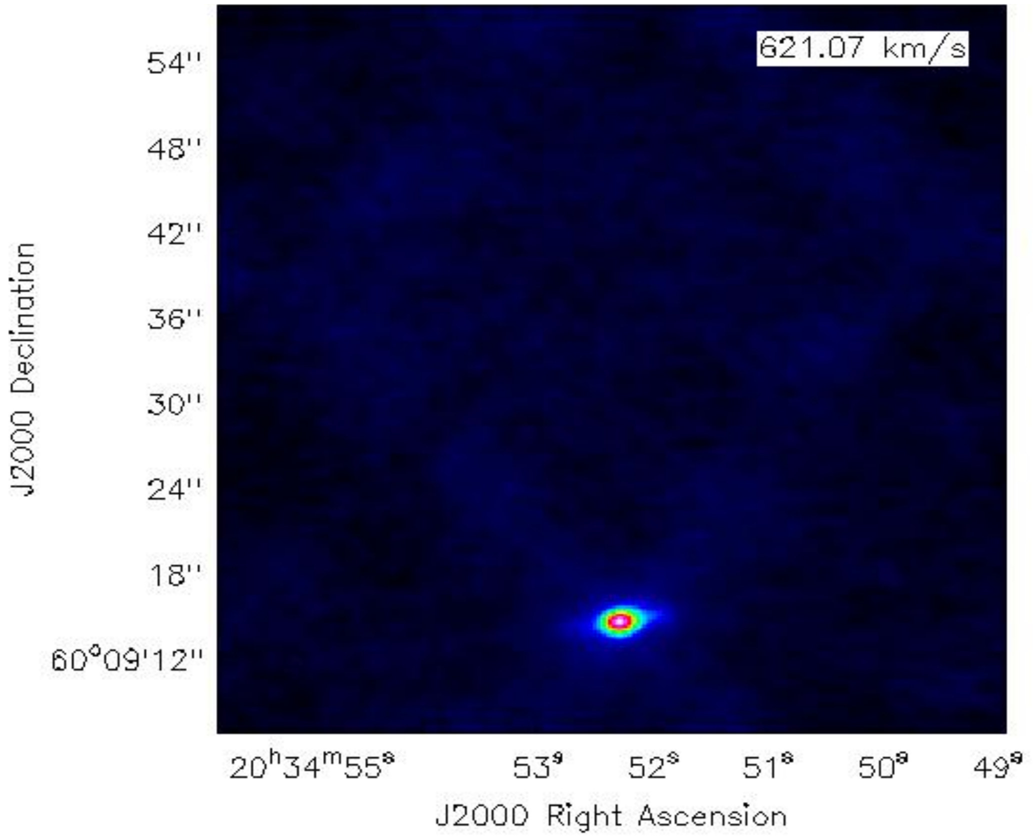
27 independent antennae (dishes), each with a diameter of 25m

NGC6946: where is the NH_3? and how cold is it?

Optical + X-ray combined

20 million light years from earth
(image about 50 arcsecs wide)

Radio: 24 GHz (λ=12.5 mm)

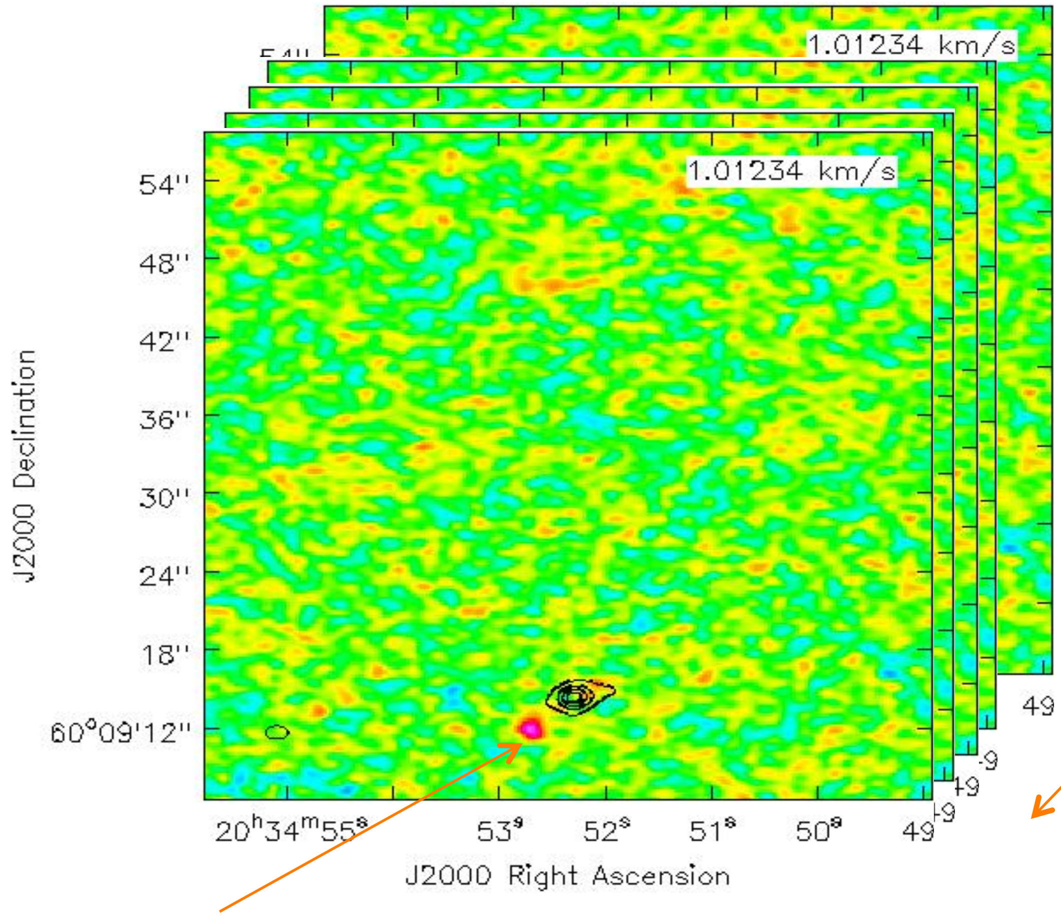


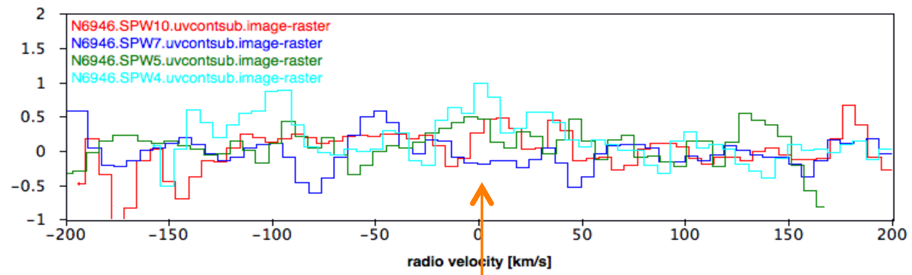
1.76 GB of “radio data”
(a few fJ in total, a few B photons)

Technische Universiteit Eindhoven University of Technology

NGC6946: where is the NH_3? and how cold is it?

„image cube”: (256 ×256 pixels) × 640 channels





19° Kelvin

640 channels of 500 kHz each

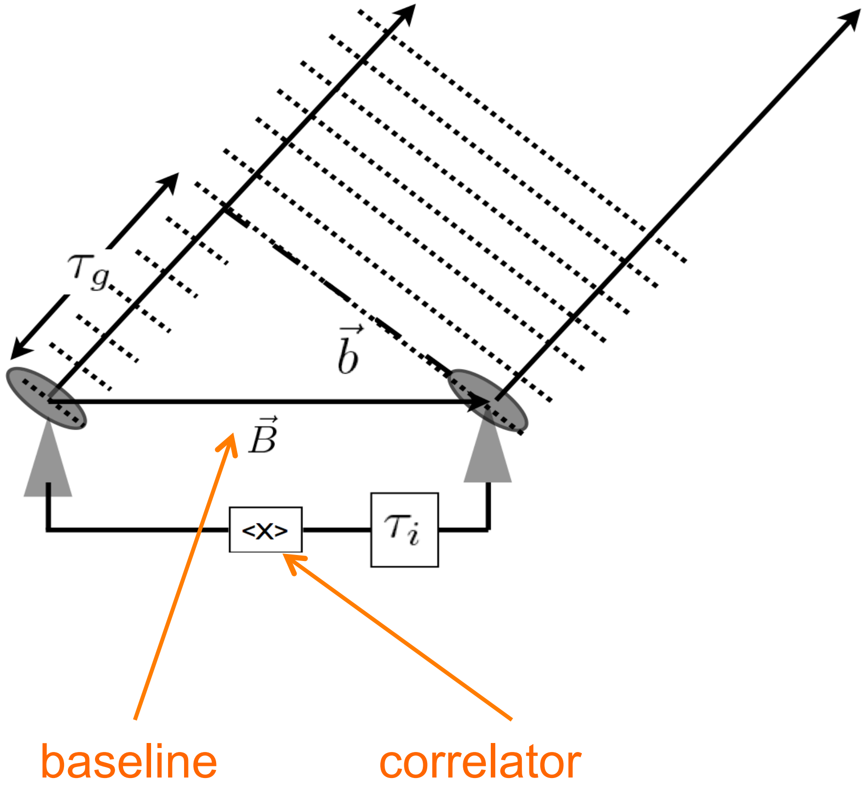
NH_3 cloud

Exascale Computing for Radio Astronomy: GPU or FPGA?

Computing:	• for the Square Kilometer Array (y2022)
• what kind is needed?	• 2D-FFT, (de-)convolution, filters, dedispersion, and a lot more
• how much?	• "exa-scale": 10^{18} FLOP/sec, i.e. 10× fastest computer existing
• in what form?	• $10^{4.5}$ nodes × $10^{4.5}$ ALUs × $f_c=10^9$ Hz?
• accelerator / node?	• GPU or FPGA?
• how to find out?	• use rooflines as a tool, for modeling and for prediction

Interferometry

2-element interferometer



Output of the correlator:

$$\mathrm{V}_\nu(\mathbf{r_1},\mathbf{r_2}) = \langle \mathbf{E}_\nu(\mathbf{r_1})\mathbf{E}^*_\nu(\mathbf{r_2})\rangle$$

- $E_v(r_1)$ is the electric field at position r_1,
- v the observation frequency, and
- * denotes complex conjugation

Van Cittert–Zernike theorem [1934-38]

correlator output

sky intensity

speed of light

$$\mathsf{V}_\nu(\mathbf{r}_1, \mathbf{r}_2) \approx \int I_\nu(\mathbf{s}) e^{-2\pi i \nu \mathbf{s} \cdot (\mathbf{r}_1 - \mathbf{r}_2)/c} \, d\Omega$$

quasi
monochromatic

base line vector,
separating the 2 antennae

solid angle

Adding geometry (assuming “narrow field”):

$$V_\nu(u, v) = \iint I_\nu(l, m) e^{-2\pi i(ul + vm)} \, dl \, dm$$

2D Fourier transform!

where *(l, m)* are sky-image coordinates
and *(u, v)* are coordinates of the base-line vector

[Tay99, Cla90, Tho01]

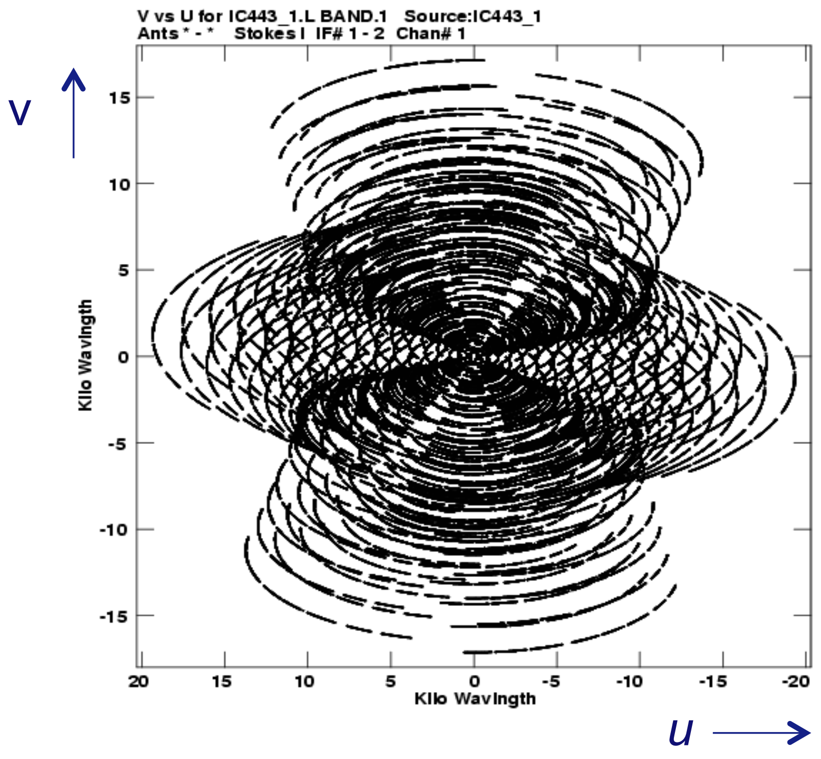
Van Cittert–Zernike theorem [1934-38]

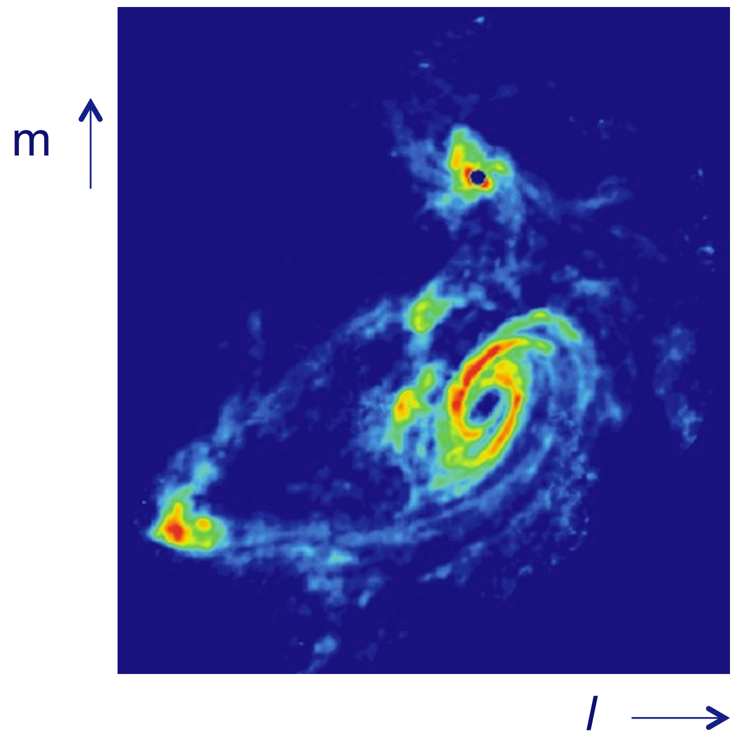
In principle:

(u, v) coverage
(A, φ) at (u,v)

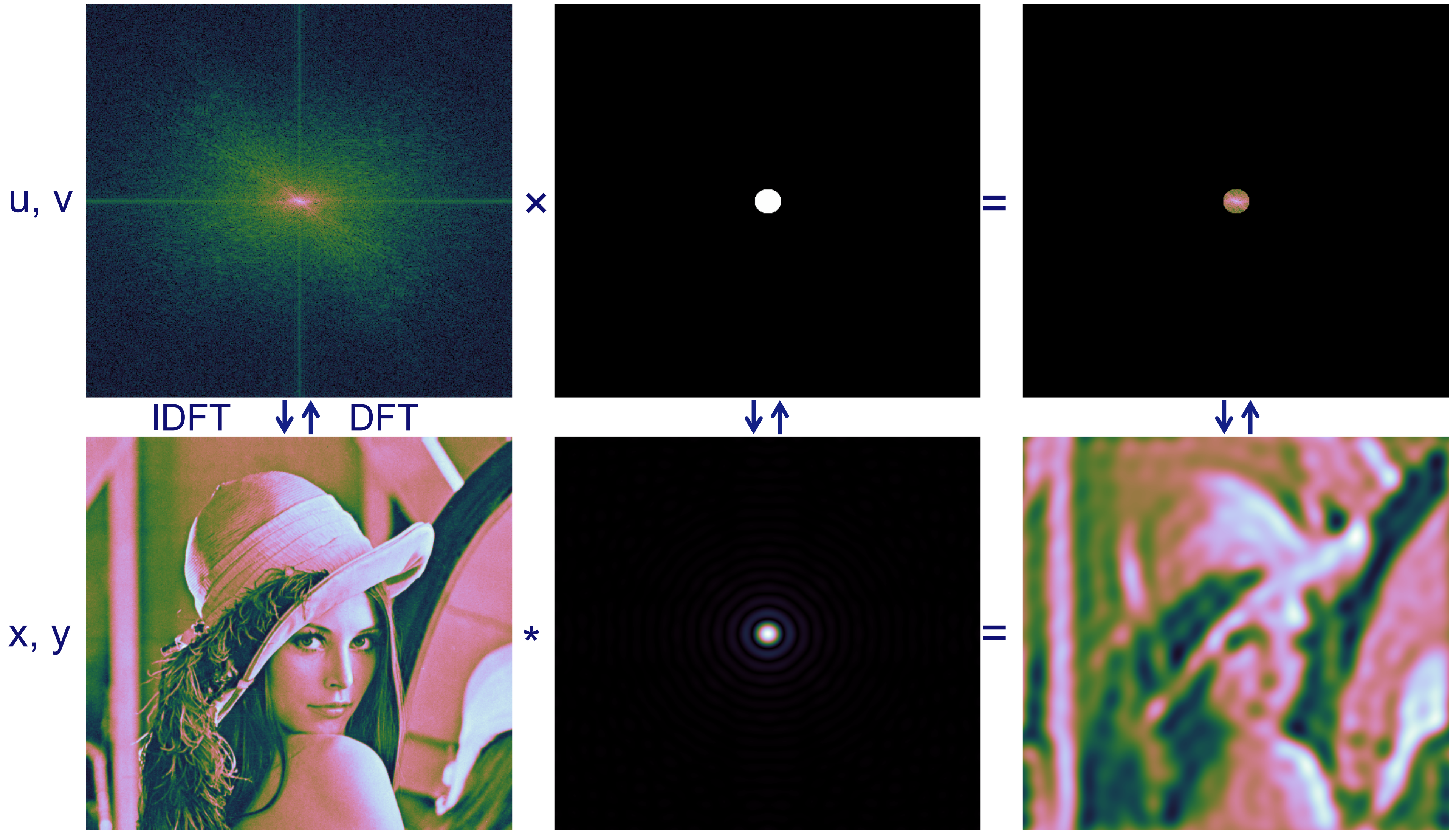
$$\begin{tikzcd} \text{(u,v) coverage} \arrow[r, bend left, "\text{I-DFT}"] & \text{(l,m) image} \arrow[l, bend left, "\text{DFT}"] \end{tikzcd}$$

(l,m) image
pixel intensity





Sampling Lucy in u-v domain with a disc



DFT convolution theorem

visibility | sampling function | observed visibility | complex (hermitian)

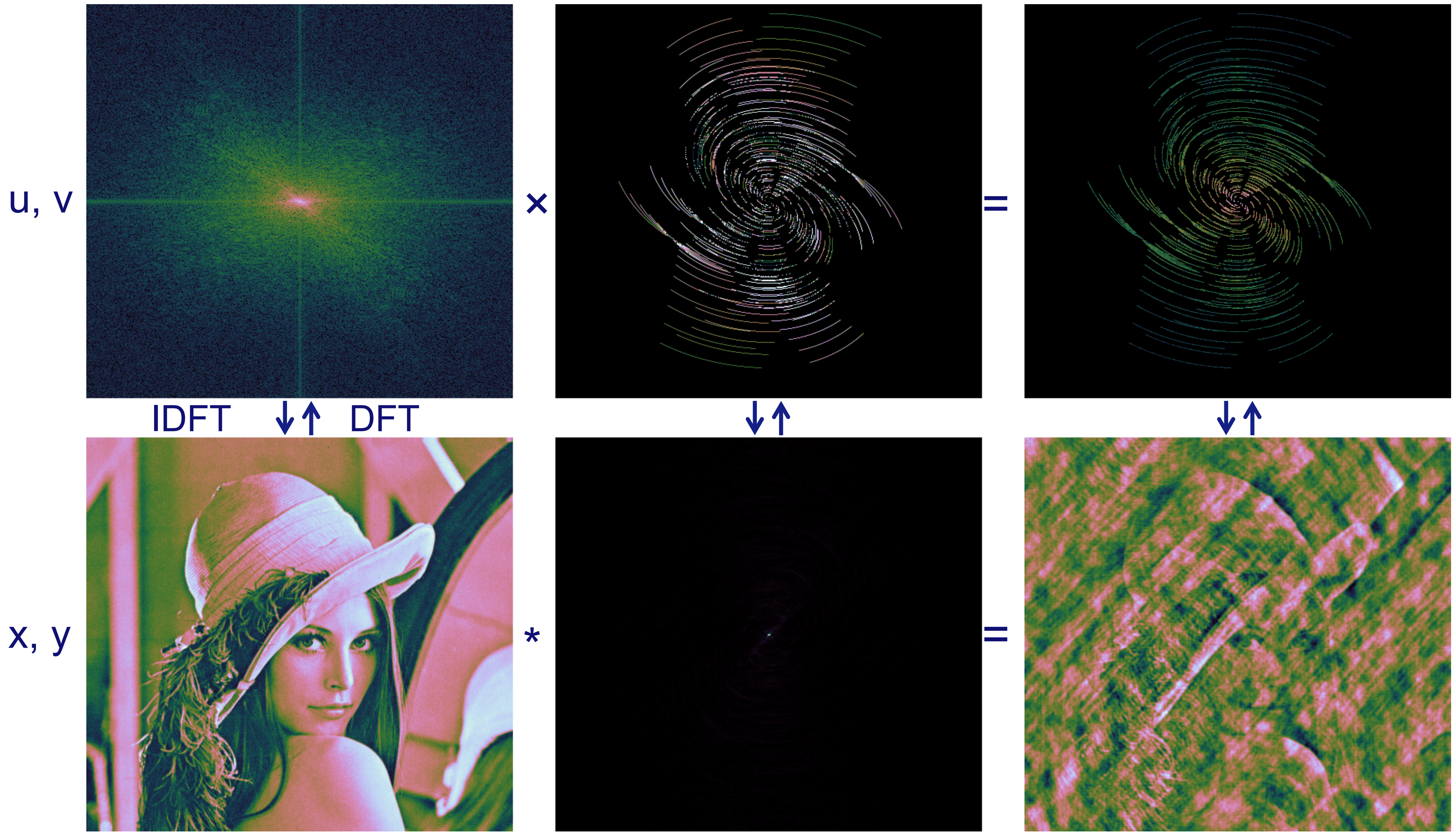
$$\begin{tikzcd}
V_\nu(u,v) \arrow[d, leftrightarrow] & \times & S(u,v) \arrow[d, leftrightarrow] & = & VS_\nu(u,v) \arrow[d, leftrightarrow] \arrow[dllll, "\text{“de-convolution”}"'] \\
I_\nu(l,m) & * & B_\nu(l,m) & = & I_\nu^D(l,m) \\
 & \text{convolution} \arrow[u] & & &
\end{tikzcd}$$

↑ DFT

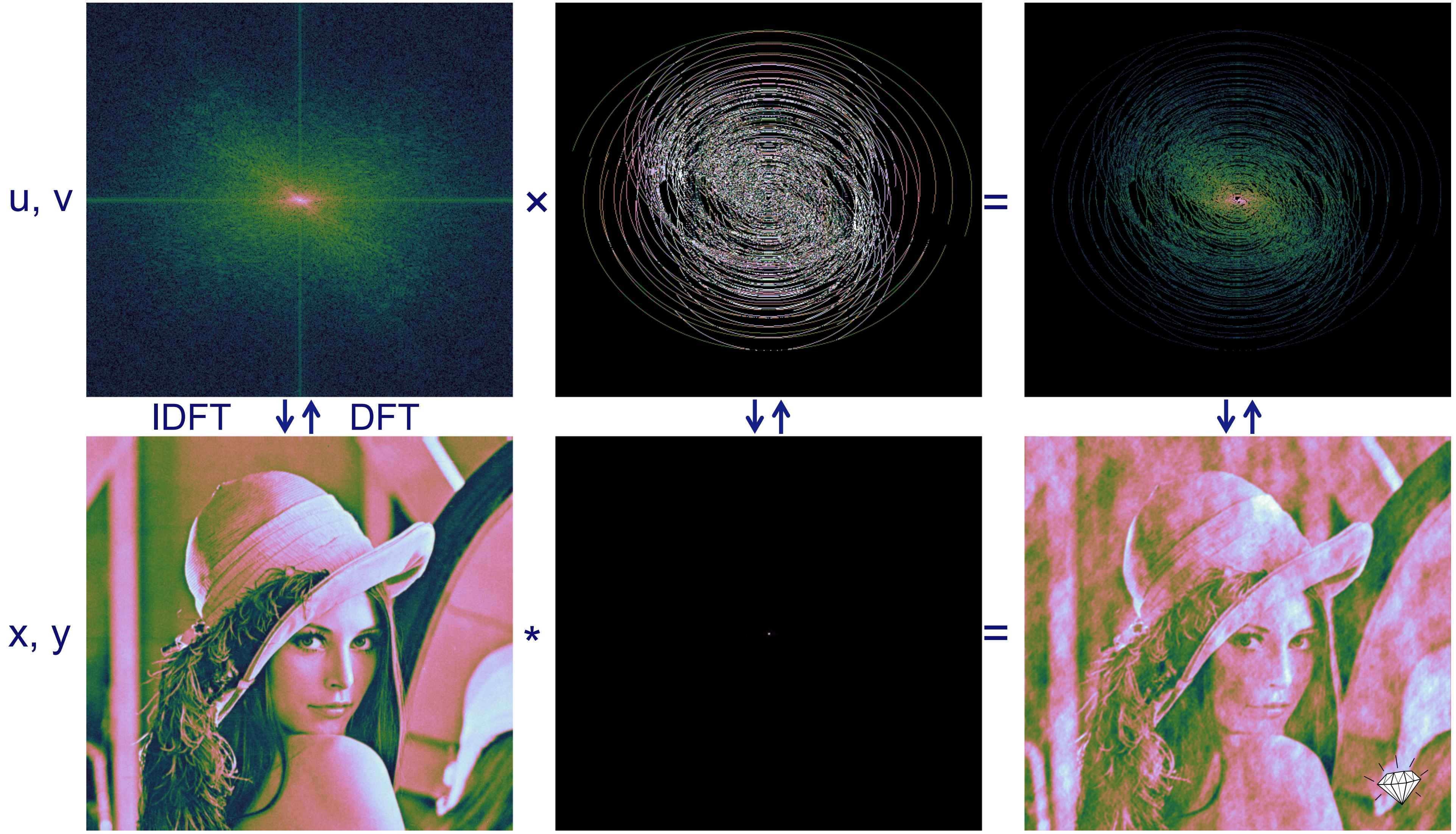
↓ I-DFT

image map | dirty beam point spread function | dirty image dirty map | real

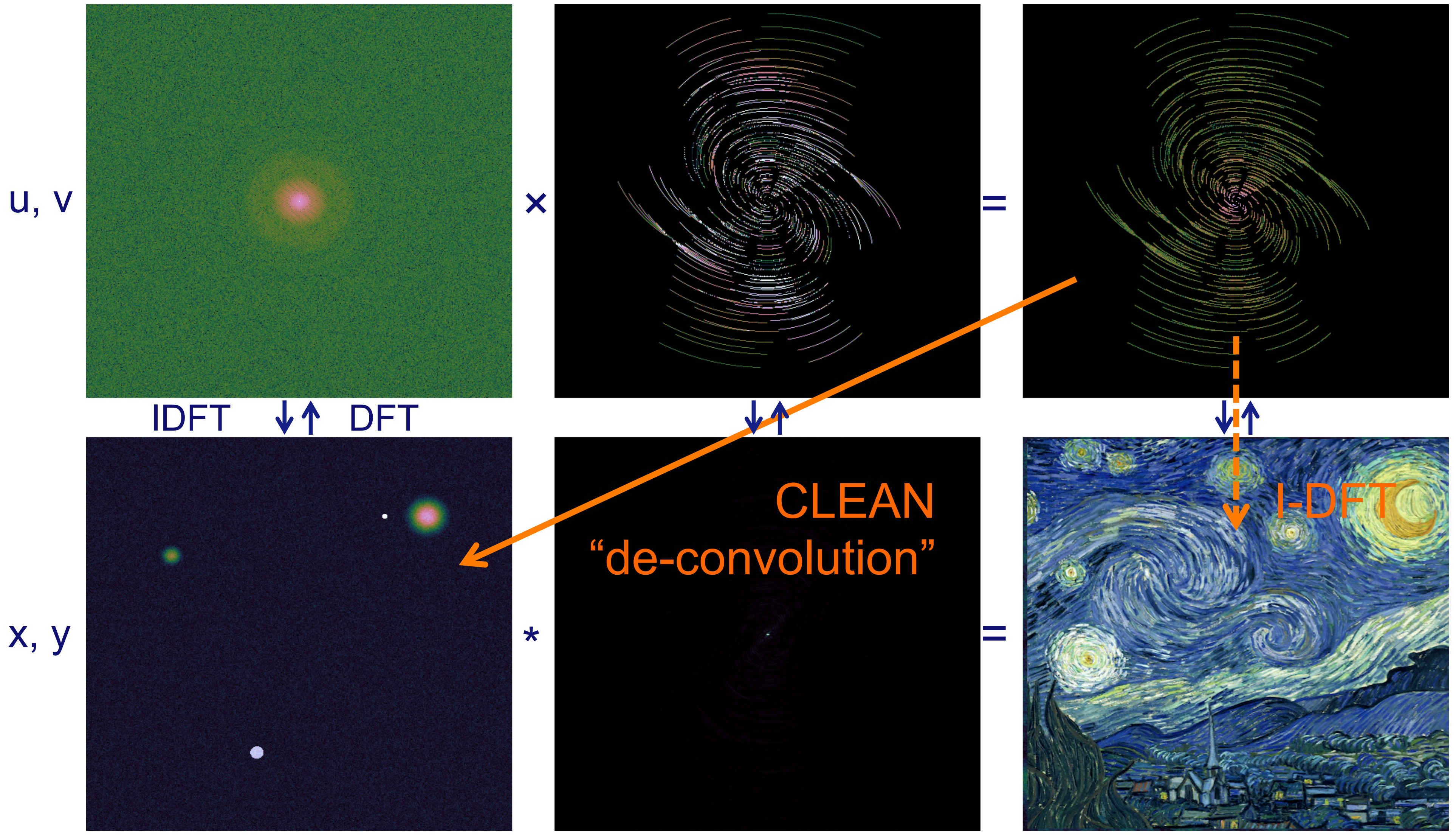
DFT convolution : Lucy with 2 hours VLA time



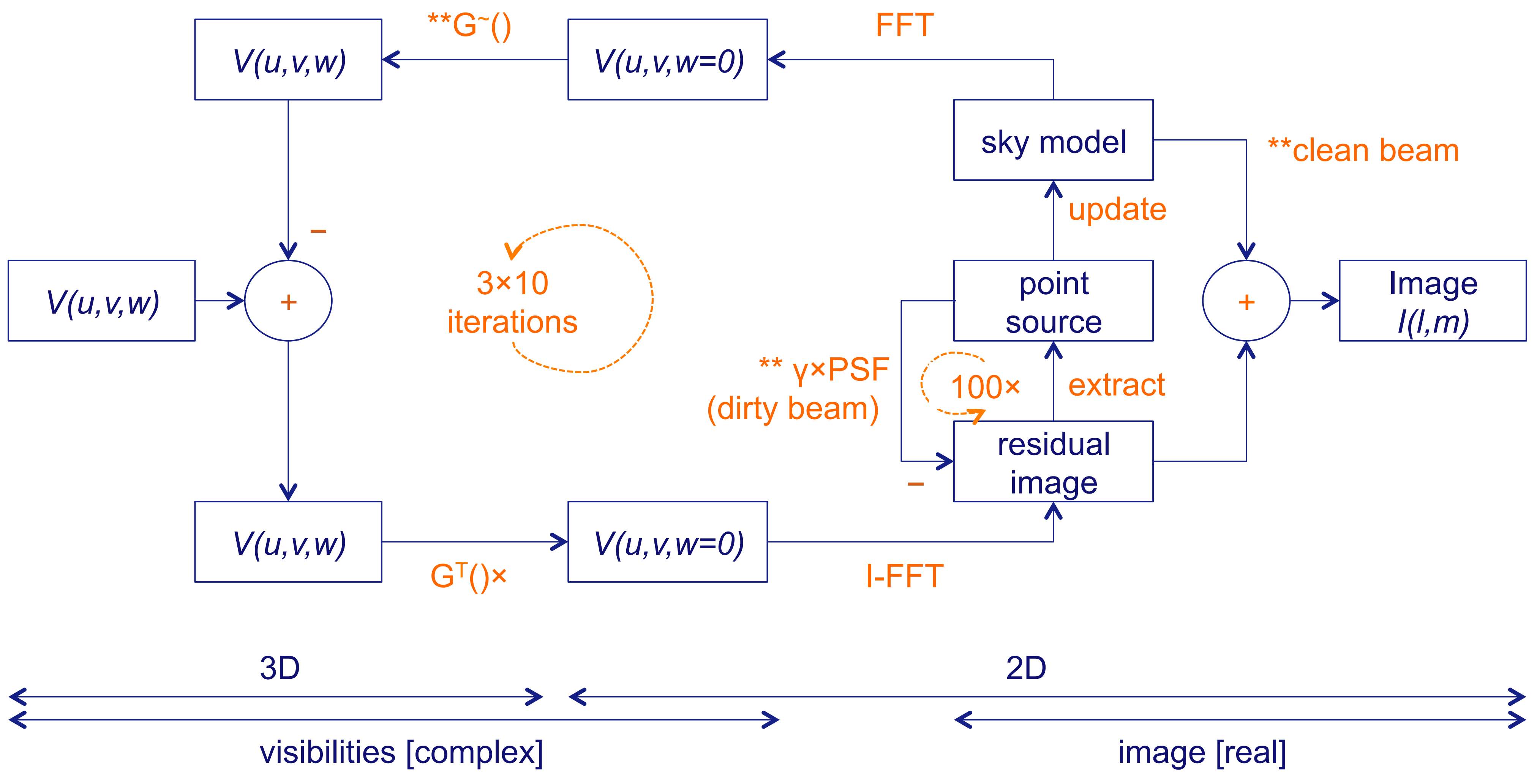
DFT convolution : Lucy with 12 hours VLA time



DFT convolution: synthetic sky with 2 hours VLA time



De-convolution (“imaging”) based on CLEAN



[Hög74]

(W-projection/snapshot implicit)

SKA1-mid [South Africa]: science in 2020

Towards a Square Kilometer Array

photograph

artist impression

SKA Organisation
/Swinburne Astronomy Productions

[Dew13]

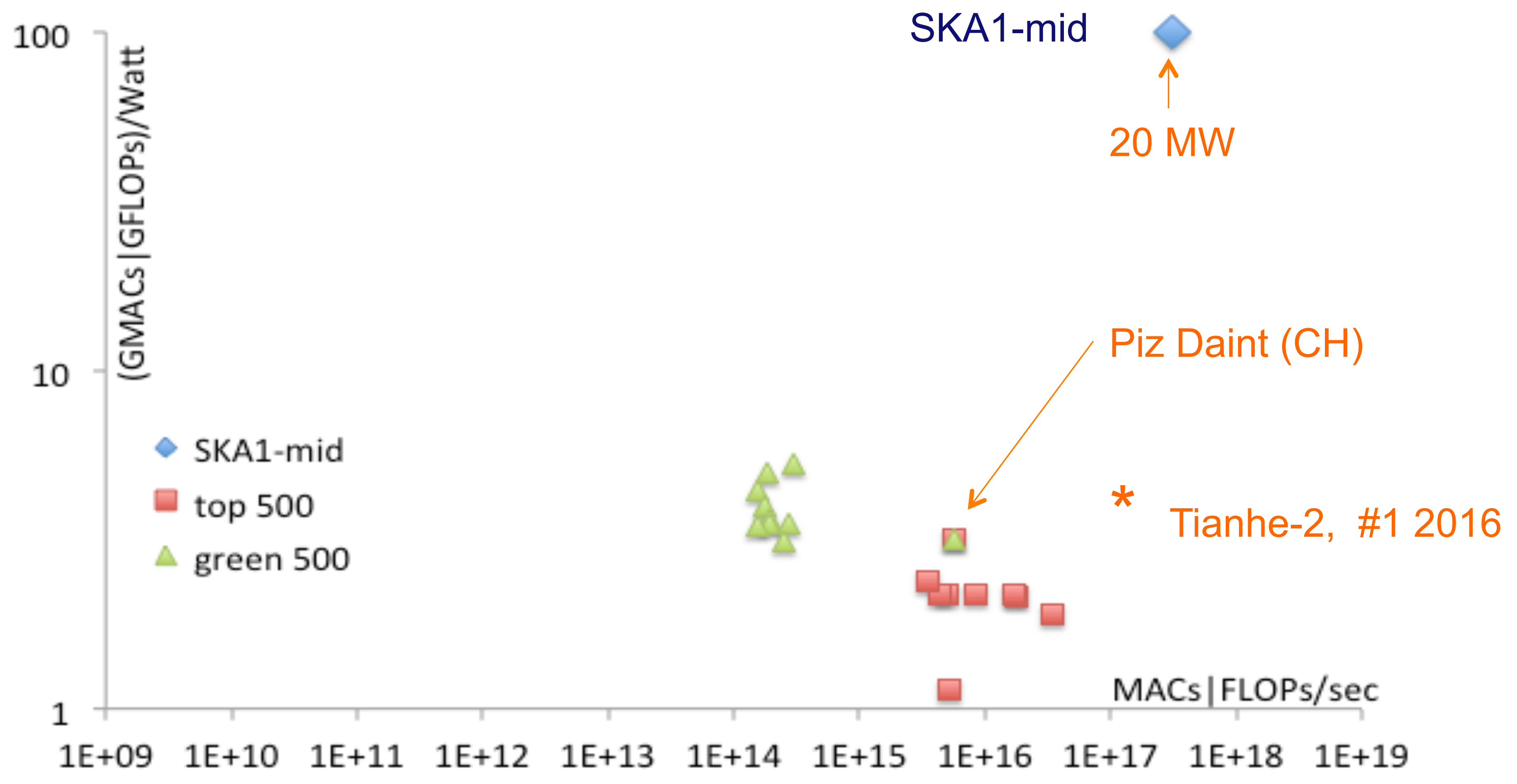
Imaging: compute load for SKA1-mid

quantity	unit	$^{10}\log$	note
# base lines		5+	$2^2 \times (\#\text{dishes})^2 = (2\times200)^2$
dump rate	s^{-1}	1+	(integration time = 0.08s) $^{-1}$
observation time	s	3	
# channels		5	"image cube" for spectral analysis
# visibilities / observation		*14.5*	*= input to imaging (≈ 10^{16} Byte)*
# op /visibility /iteration		4.5	convolution, matrix multiply, (I)FFT
# major iterations		1.5	(3×calibration) × (10×major)
# op /observation		*20.5*	
# op /sec	*Hz*	*17.5*	≈ 1 exaflop/ sec

- #operations/visibility/iteration depends on *W*-projection method
- calibration loop (3×) around imaging loop

[Jon14, Ver15, Wijn14]

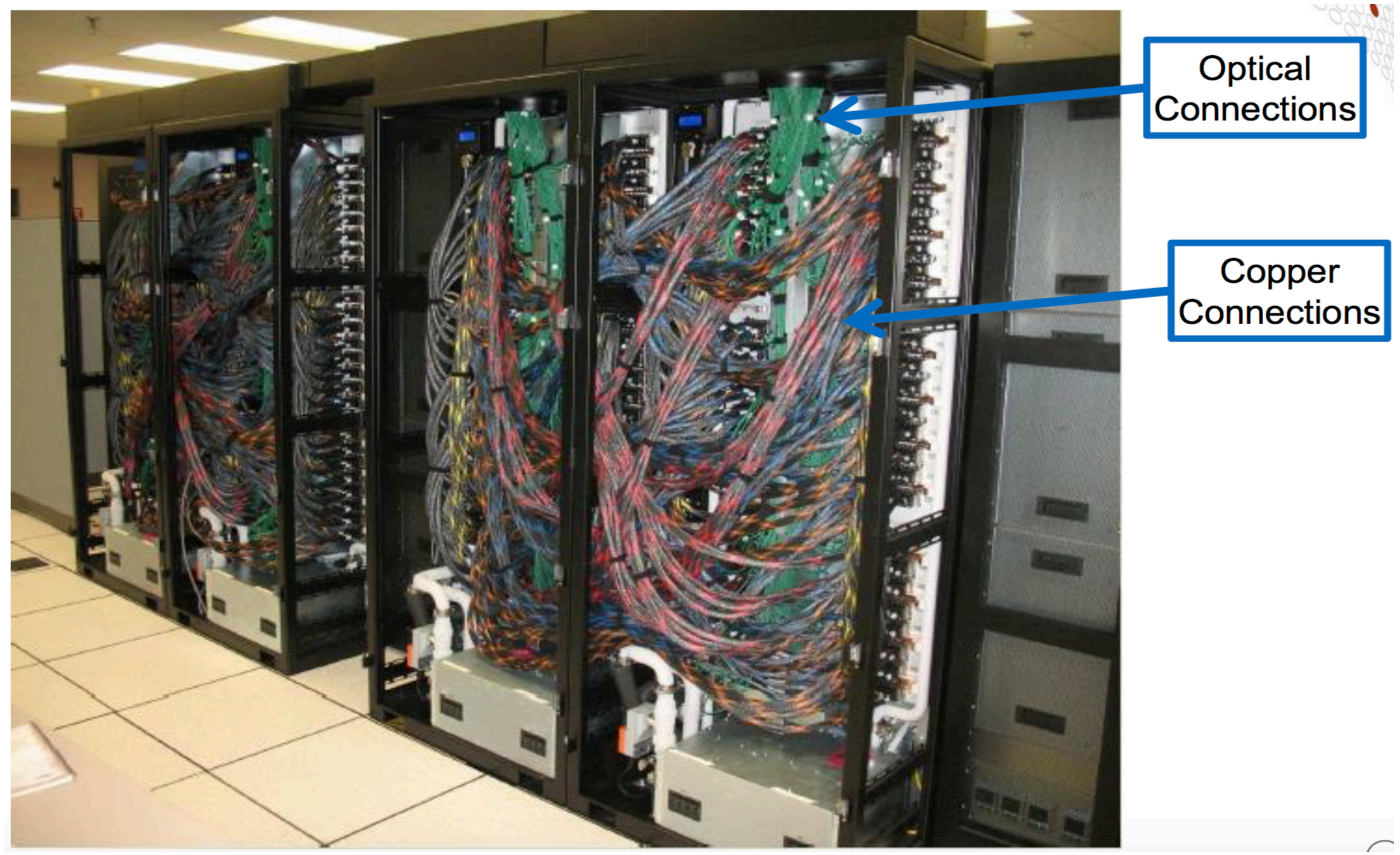
EXAflops/sec in 2015?



- *net* SKA1-mid computation load "*2020*" versus
- *gross* (peak) compute performance "*2015*"

[Gre14]

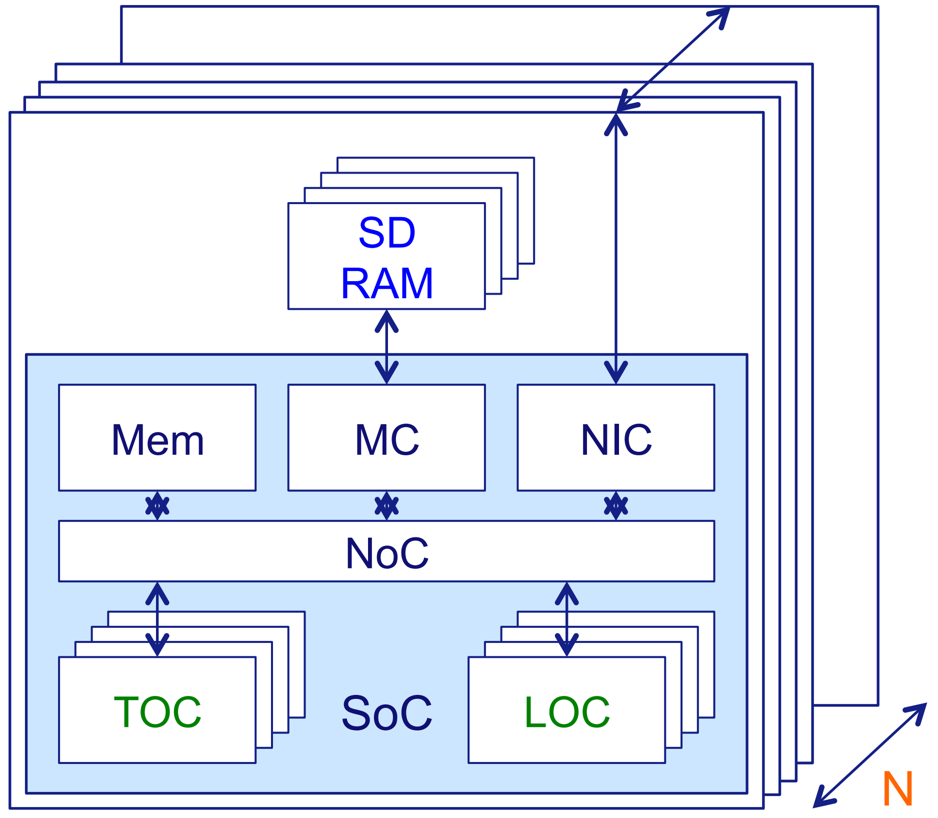
Piz Daint: a Cray XC30 system



Comprising many kilometers of (optical) cable, ... and 5272 nodes

Super computers 101

A modern supercomputer = *N* (10^3 - 10^5) identical nodes connected by a network (ignoring storage, peripherals, service nodes, …)



“TOC, LOC” is Nvidia speak

network (system-level interconnect)

Synchronous DRAM / node

Mem	= on-chip memory, e.g. L2
MC	= Memory Controler
NIC	= Network Interface
NoC	= Network on Chip
TOC	= Throughput Optimized Core
LOC	= Latency Optimized Core
SoC	= System on Chip

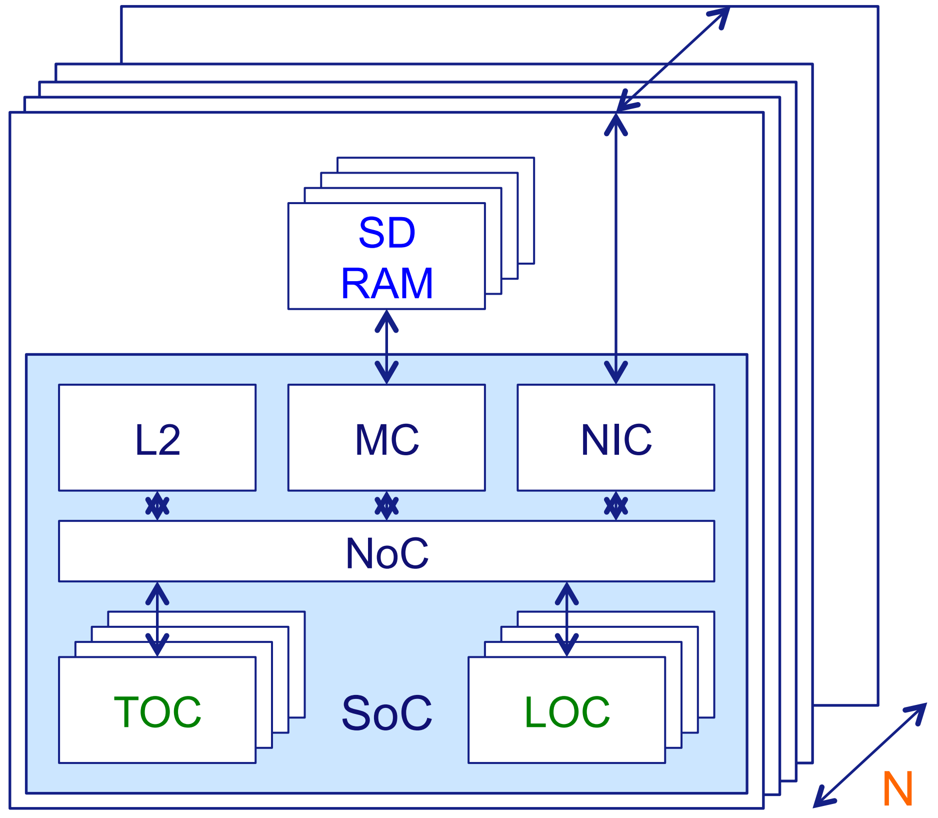
Piz Daint: 7.8 Pflops/s @ 1.8MW

Cray CX30, N= 5272

TOC = Tesla K20X GPU

LOC = 8× Intel Xeon @2.6 GHz

Aries network, Dragonfly router IC

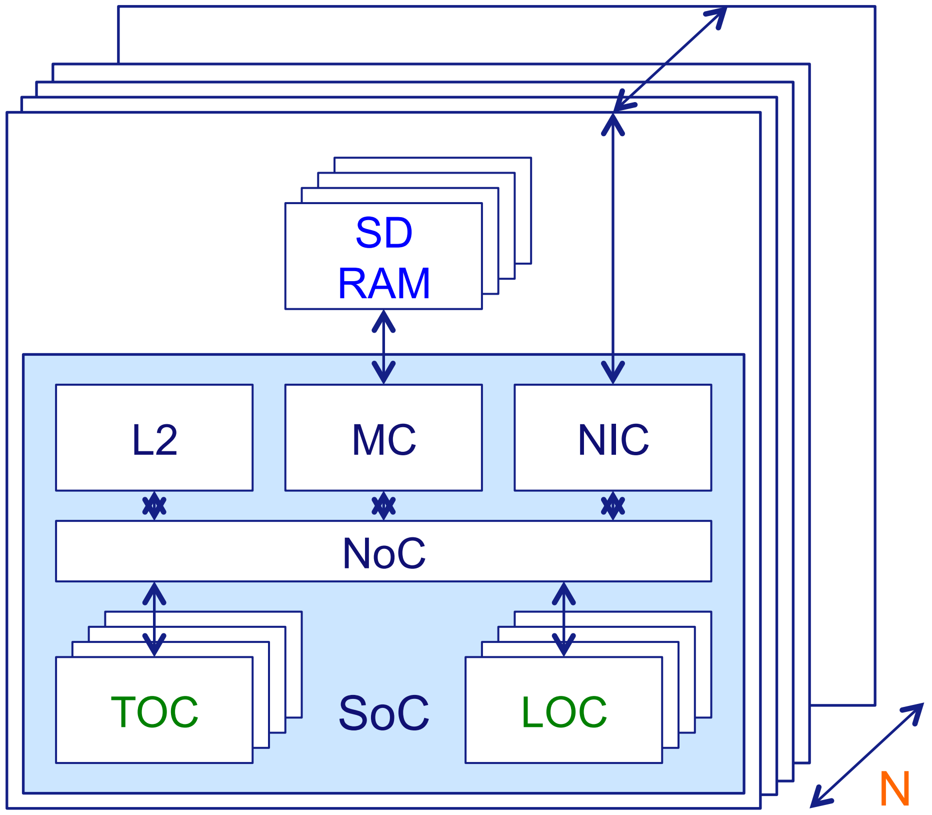


Piz Daint	node	system
# nodes	1	5272
Xeon 2.6GHz	0.17	
Tesla K20X	1.31	
TFLOP/s	1.48	7787
TB	0.06	337
kW	0.33	1754

Nvidia Research [Ore14]: 1 ExaFlops/s @ 23MW in 2020

N= 76800 (7nm CMOS) [Ore14]

TOCs = 8,192 multiply-add @ 1GHz double-precision

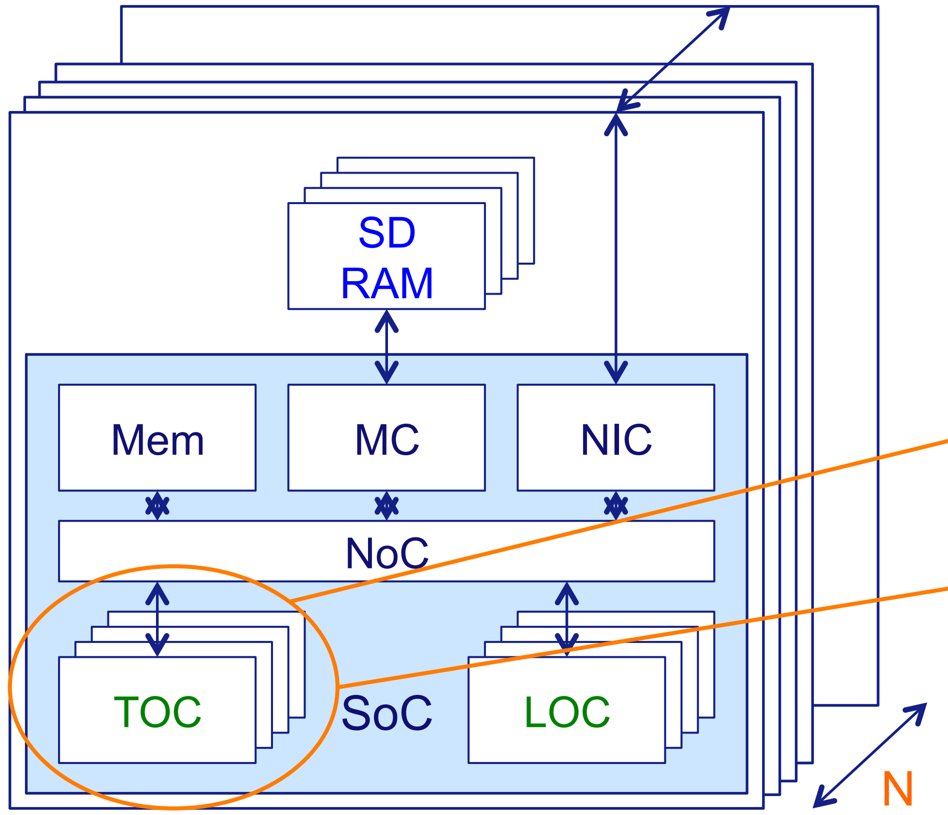


Aries-like network

Nvidia 2020		node	system
# nodes		1	76800
TOC		16.4	
TFLOP/s	PF/s	16.4	1258
TB		0.51	39322
kW		0.30	23000

Exascale computing for radio astronomy

Exascale computing: 10^{18} flops



Radio astronomy: $10^{17.5}$ flops

with gridding (*W*-projection)
and 2D-FFT as heavy kernels.

Let's map 2D-FFT on a node.

Option 1: FPGA*

Option 2: GPU

* in same package (not same SoC)

State-of-the-art GPU and FPGAs

		Nvidia GP100	Intel/Altera Stratix 10	Xilinx VU13P
cmos	nm	16	14	16
clock frequency	MHz	1328	800	*800
scalar/dsp processors		3584	11520	11,904
peak throughput	GFLOP/s	9519	9216	7619
data type [32b]		float	float	fixed
DRAM interface		HBM2	#HBM2	#HBM2
DRAM bandwidth	GB/s	256	256	256
power consumption	W	300	126	
GfFLOP/W		32	73	

*assumption, no data found
#HBM2 (High Bandwidth Memory) interface to 3D stacked DRAM is an option.

DFT in matrix-vector form

Let *x* and *X* be *complex* vectors of length *N.*

$$X^T = F_N \cdot x^T \qquad or \qquad (X_0, X_1, X_{N-1})^T = F_N \cdot (x_0, x_1, x_{N-1})^T$$

Where F_N is the twiddle factor matrix, $\omega = e^{2\pi i/N}$

$$F_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad F_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}, \quad F_n = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega & \cdots & \omega^{n-1} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & \omega^{n-1} & \cdots & \omega^{(n-1)^2} \end{pmatrix}$$

In 2 dimensions: $Y = F_M \cdot X \cdot F_N$

Where *Y* and *X* are matrices of size *M×N.*

$F_M \cdot X$: apply *M*-point 1D-DFT to each column of matrix *X*.

DFT: Cooley-Tukey factorization theorem

Let $N=P\times M$. Then F_N can be factorized as [Loa92]

$$F_N = P_{N,P}\boxed{(I_P \otimes F_M)}\boxed{D_N}\boxed{(F_P \otimes I_M)}$$

Where

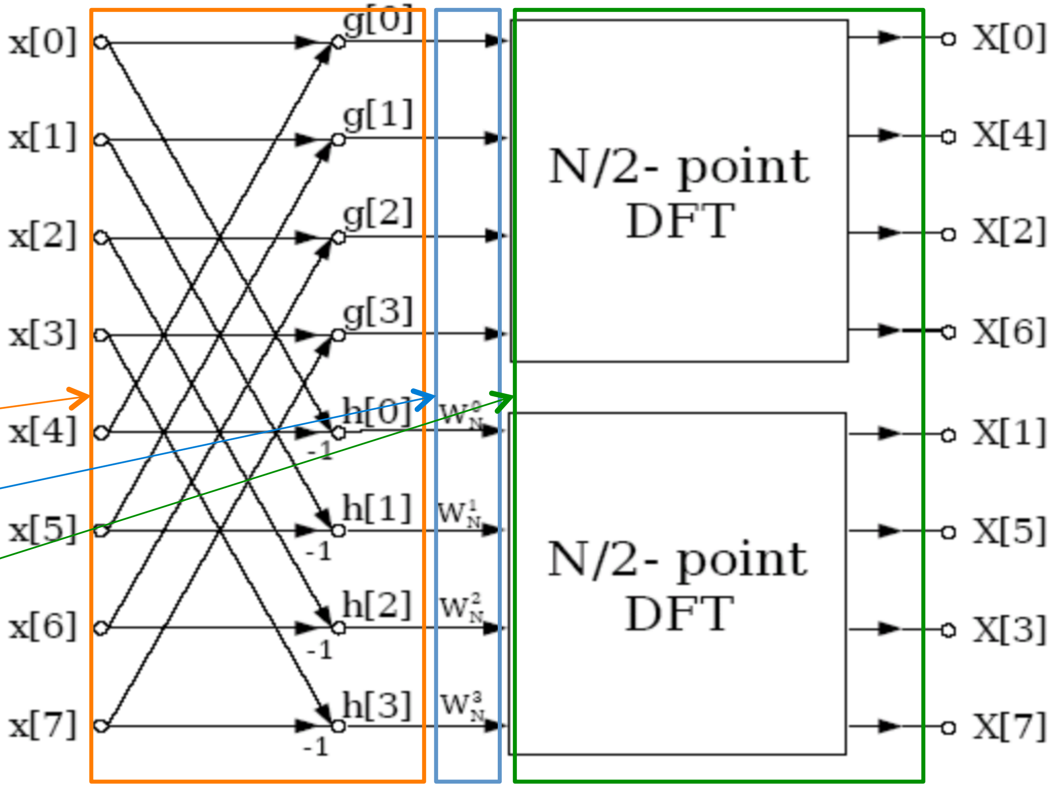
I_M and I_p are identity matrices,

$\otimes$ denotes the tensor product,

D_N the diagonal twiddle matrix.

Example: $P=2$, $M=N/P=4$:

- $\boxed{M \text{ radix-2 DFTs}}$
- $\boxed{D_N}$
- $\boxed{P \text{ radix-}N/2 \text{ DFTs}}$
- $P_{N,P}$ omitted



Cooley Tukey factorization is the basis of FFT

2D-FFT: arithmetic intensity

The arithmetic intensity I_A = amount of compute per unit problem size

$$I_A = \frac{number\_of\_operations}{size\_of\_(input + output)\,[bytes]}$$

For a 2D-FFT of size $N{\times}N$ with complex input and output we have:

$$I_A(N) = \frac{2 \times N \times \left[\boxed{{}^{1}\!/_{2}\, N \log_2(N)\; butterflies}\right] \times (10\; ops/butterfly)}{(1 read + 1 write) \times (N^2 pixels) \times (8\; bytes/pixel)}$$

$$I_A(N) = 0.625 \log_2(N) \qquad ops/byte$$

With $2^{10} \leq N \leq 2^{14}$ this amounts to $6.25 \leq I_A(N) \leq 9.38$.

2D-FFT: operational intensity

The arithmetic intensity I_A = amount of compute per unit problem size

$$I_A = \frac{number\_of\_operations}{size\_of\_(input + output)\,[bytes]}$$

The operational intensity I_{OP} = amount of compute per unit DRAM traffic

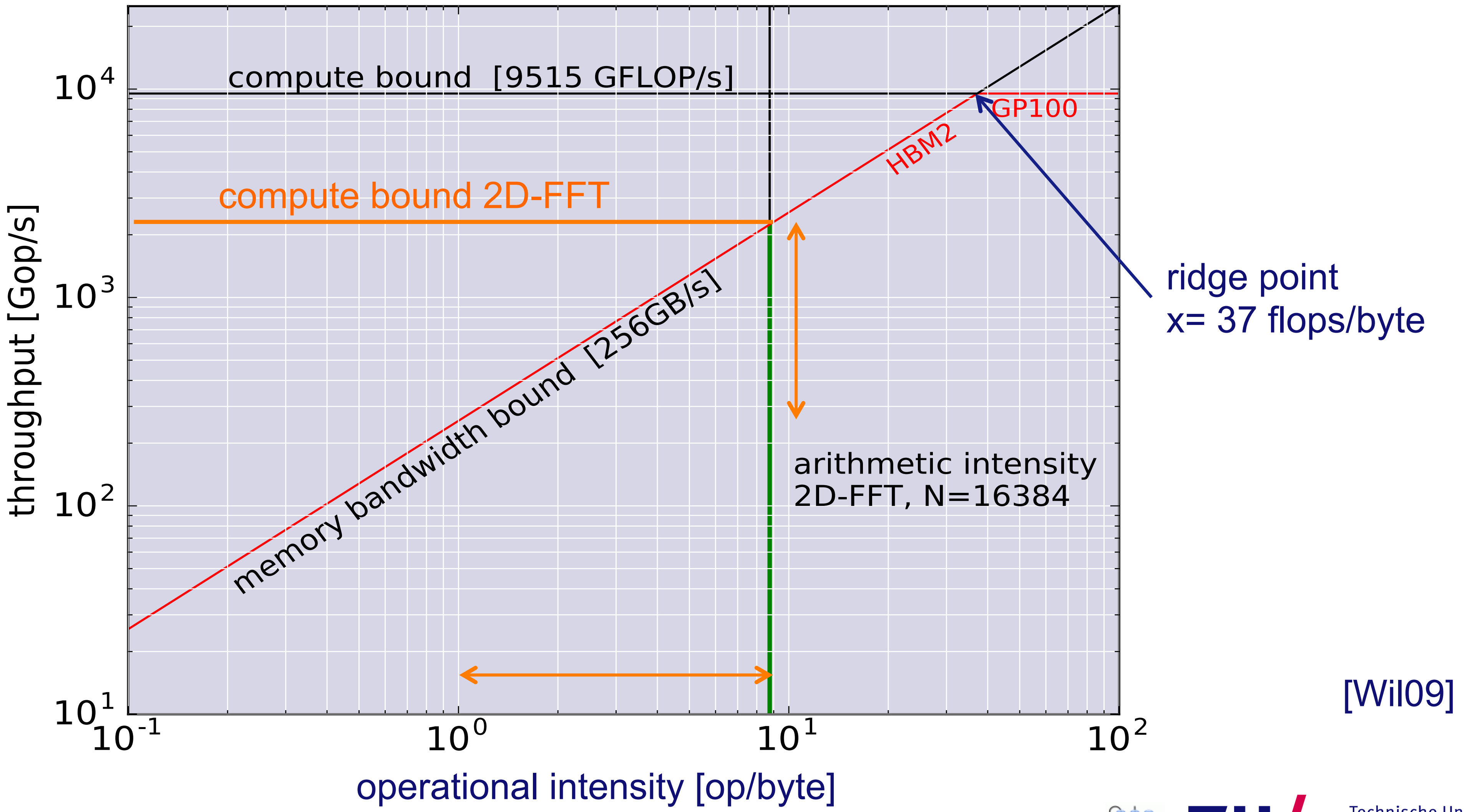
$$I_{OP} = \frac{number\_of\_operations}{amount\_of\_DRAM\_traffic\,(input + output)\,[bytes]}$$

[Wil09]

$I_{OP} = I_A$ only if entire problem fits in on-chip memory.

In practice $I_{OP} << I_A$
and depends on algorithm choices and on available on-chip memory.

Roofline = compute and memory bandwidth bounds



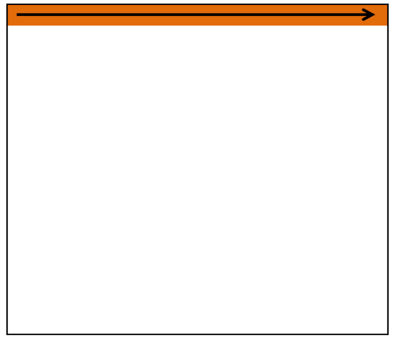
[Wil09]

2D-FFT: “classical” row-column algorithm

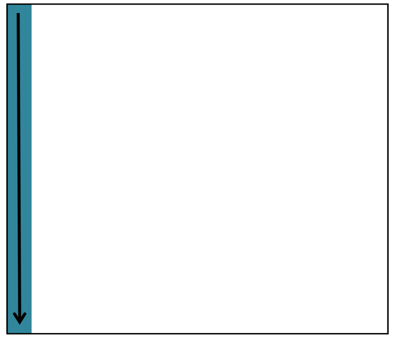
1. apply 1D-FFT to individual rows

2. apply 1D-FFT to individual columns

pass 1



pass 2



During 2.: with DRAM transaction size =*B* pixels,
B-1 pixels are read/written without being used.
If *B>1* then memory bandwidth under utilized.

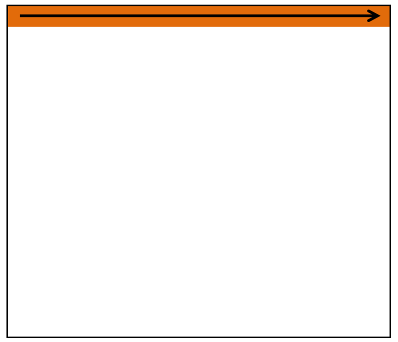
1+*B* read-write passes to DRAM, hence:

$$I_{op,row-col}(N) = \frac{1}{1+B} I_A(N) \quad << \quad 0.31\log_2(N) \quad ops/byte$$

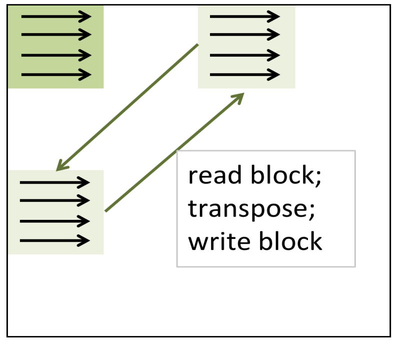
2D-FFT, using matrix transposition

1. apply 1D-FFT to individual rows;
2. transpose matrix block by block (size *B*×*B*) in on-chip memory;
3. apply 1D-FFT to individual transposed columns;
4. transpose matrix.

pass 1

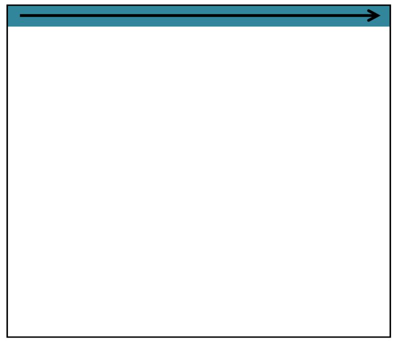


pass 2, 4



On-chip memory: *2×max (B×B, N)* pixels

pass 3



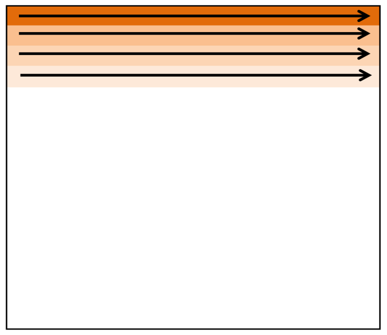
4 read-write passes to DRAM, hence:

$$I_{op,\,transpose}(N) = \tfrac{1}{4} I_A(N) = 0.16 \log_2(N) \quad ops\,/\,byte$$

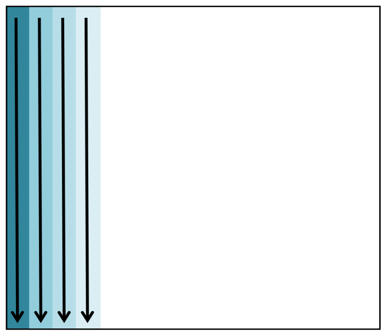
2D-FFT by processing *B* rows/columns in ||

1. apply 1D-FFT to *B* rows rows in ||
2. apply 1D-FFT to columns in ||

pass 1



pass 2



On-chip memory: (±2) × *B* × *N* pixels

2 read-write passes to DRAM, hence:

$$I_{op,B-row-col}(N) = \tfrac{1}{2} I_A(N) = 0.31 \log_2(N) \quad ops/byte$$

2D-FFT by processing *B* segmented columns in ||

Columns: Cooley-Tukey factorized into 1b +2

1. a) apply 1D-FFT to N_R rows in ||
   optimal: √B rows
   b) apply partial 1D-FFT
   to N_C columns in ||

2. apply partial 1D-FFT
   to column segments in ||

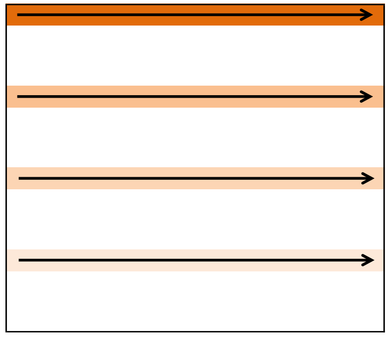
On-chip memory: (±2) × *max*(N_R, √B) × *N* pixels

2 read-write passes to DRAM, hence:

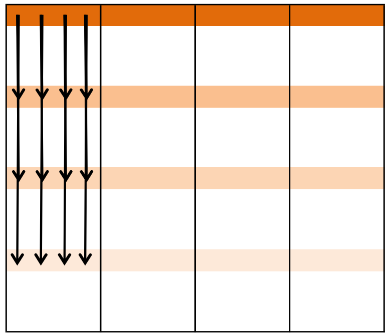
$$I_{op,segm-col}(N) = \tfrac{1}{2} I_A(N)$$
$$= 0.31 \log_2(N) \quad ops / byte$$

[Yu10]

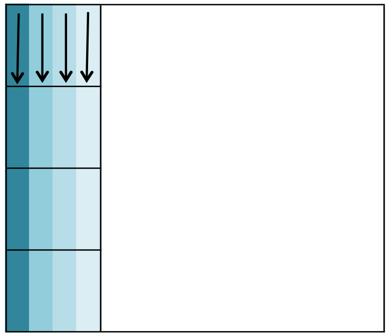
pass 1a



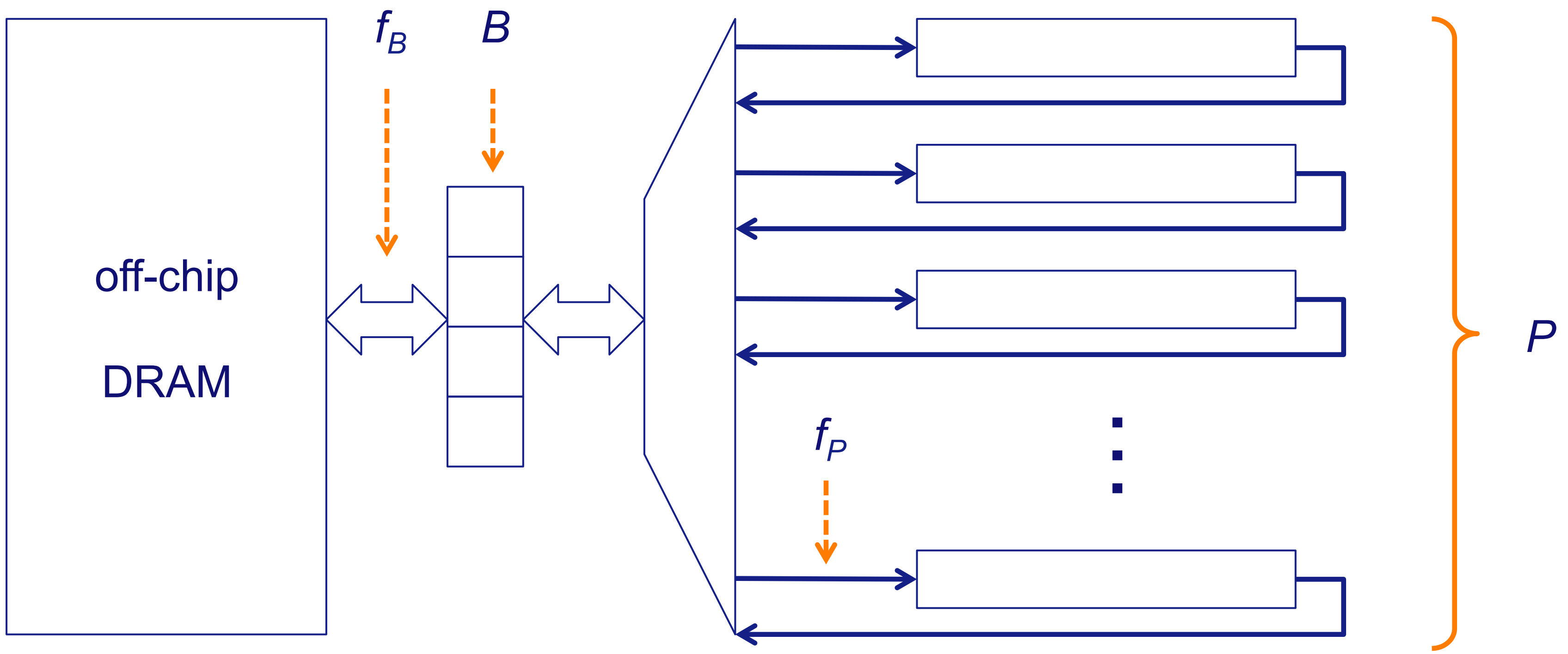
pass 1b



pass 2



2D-FFT on FPGA, based on pipelined 1D-FFT



DRAM transactions (read|write)
of size *B* pixels [8Byte]
at rate f_B transactions/sec

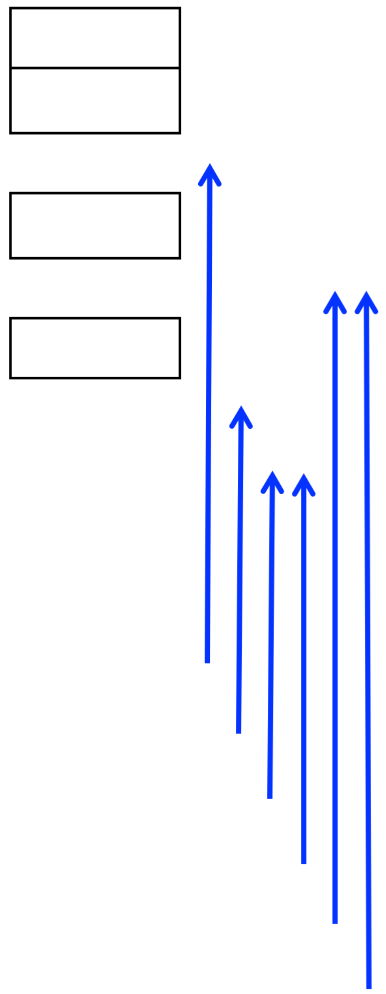
P 1D-FFT pipelines
with i/o rates of f_P pixels/sec

Rate matching eqn: $$f_B \times B = 2 \times f_P \times P$$

2D-FFT on FPGA: dimensioning

B	DRAM transaction size (max burst)	[pixel=8B]
f_B	transaction rate	[MHz]
P	numer of 1D-FFT pipelines of size N	
f_P	pixel rate per pipeline	[MHz]
M	on-chip memory	[kpixel=8kB]
N	image side, image= $N\times N$	
N_R	number of rows processed in ||	
N_C	number of columns processed in ||	

rate-matching constraint	$2\times P\times f_P \geq B\times f_B$
hence	$P > (B\times f_B)/(2\times f_P)$
parallelism constraint	$N_R \geq P$
	$N_C \geq P$
DRAM transaction constraint	$N_C \geq B$
on-chip memory constraint	$M \geq 3\times max(N_R, N_C)\times N$
alternative, segmented columns	$M \geq 3\times max(N_R, min(\sqrt{B}, N_C))\times N$



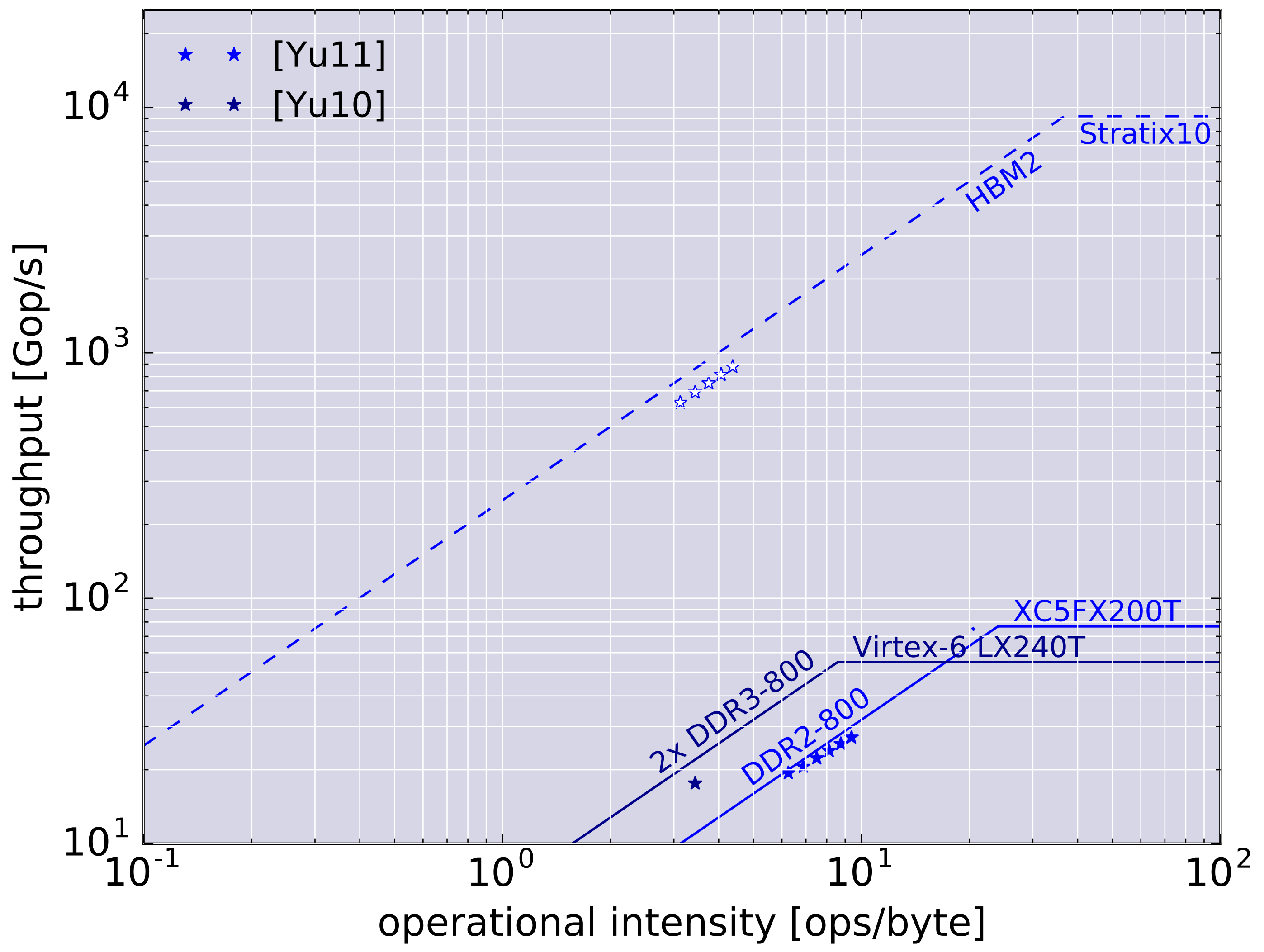
2D-FFT on FPGA: dimensioning

			[Yu10] DDR3	Stratix10 HBM2
B	DRAM transaction size (max burst)	[pixel=8B]	32	32
f_B	transaction rate	[MHz]	25	1000
P	numer of 1D-FFT pipelines of size N		5	24
f_P	pixel rate per pipeline	[MHz]	80	800
M	on-chip memory	[kpixel=8kB]	68	1152
N	image side, image= $N \times N$		4096	16384
N_R	number of rows processed in ||		5	24
N_C	number of columns processed in ||		32	24

		[Yu10] DDR3	Stratix10 HBM2
rate-matching constraint	$2 \times P \times f_P \geq B \times f_B$		
hence	$P > (B \times f_B)/(2 \times f_P)$	5	20.0
parallelism constraint	$N_R \geq P$	5	24
	$N_C \geq P$		24
DRAM transaction constraint	$N_C \geq B$	32	
on-chip memory constraint	$M \geq 3 \times \max(N_R, N_C) \times N$	384	1152
alternative, segmented columns	$M \geq 3 \times \max(N_R, \min(\sqrt{B}, N_C)) \times N$	68	1152

Technische Universiteit **Eindhoven** University of Technology

2D-FFT on FPGAs



Stratix10:
32b floating point;
throughputs based on I_{op}, 20% margin.

[Yu11]:
16b fixed point;
hence I_{op} 2×

[Yu10]:
32b fixed point

Technische Universiteit Eindhoven University of Technology

2D-FFT on a GPU

Based on [Won10], 2010:
“Demystifying GPU microarchitecture through micro-benchmarking”

Nvidia GTX200, Tesla microarchitecture:

- 30 Streaming Multi processor (SM)
- each SM contains 8 Scalar Processors (SP)
- each SP: 1 fused-multiply-add per clock cycle @ 1.35 GHz

- unit of execution flow in the SM is the *warp* comprising *32 threads*
- *“6 warps (192 threads) needed to hide register read-after-write latencies”*

- register file: 64 kB per SM (max 128 registers per thread)
- register files combined: 2MB,
  exceeding on-chip “shared memory” (by 4x) and on-chip caches!

2D-FFT on a GPU

Based on MicroSoft 2008 paper [Gov08, about 300 citations]:
"High Performance Discrete Fourier Transforms on Graphics Processors".

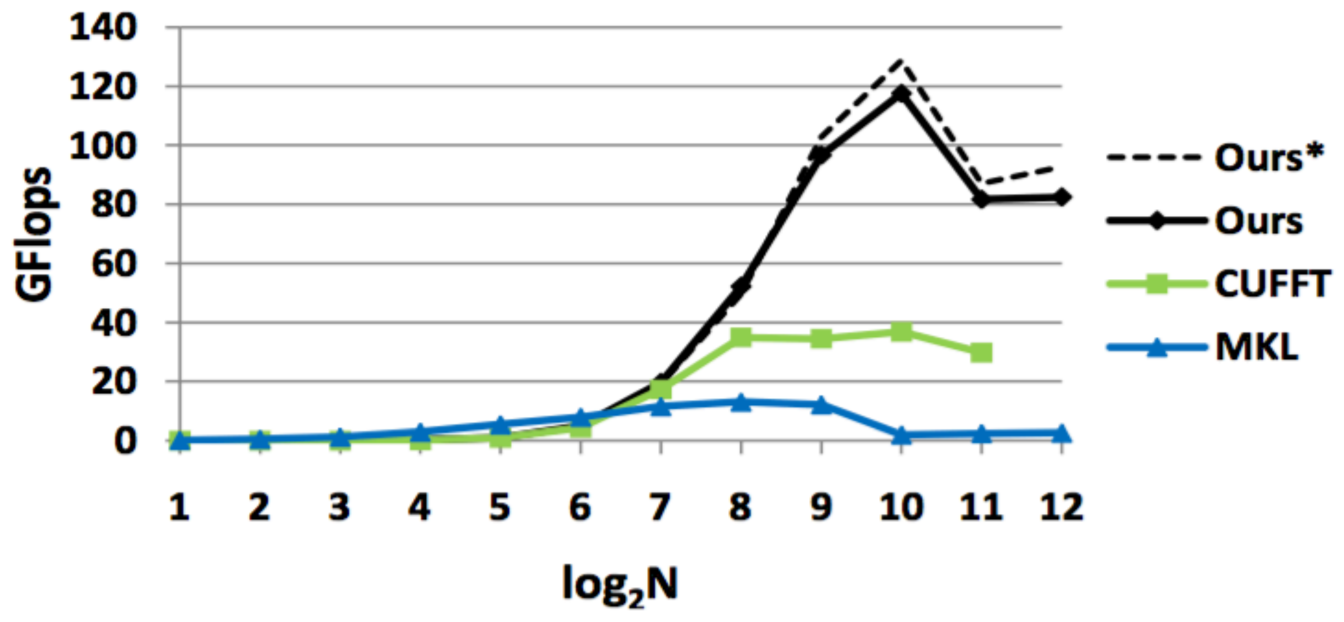
Parallelism: 1 thread = 1 butterfly!
"To maximize the reuse of data read from DRAM …, it is best to use a large radix R. However, R is limited by the number of registers and the size of the shared memory on the multiprocessors… We use R=8".

With R=8, and N=4k, "only" 4k/8 threads per 1D-FFT stage.
Hence, process M FFTs in parallel *"to achieve full utilization of the SMs or to hide memory latency while accessing DRAMs."*

After each radix-8 stage, result is written back into the off-chip DRAM:

$$I_{op,\,R8-stage}(N) = \frac{I_A(N)}{2\lceil \log_8(N) \rceil} = \frac{0.625 \log_2(N)}{2\lceil \log_8(N) \rceil} = \pm 0.87\ ops/byte$$

Measured 2D-FFT throughput on GTX280 GPU



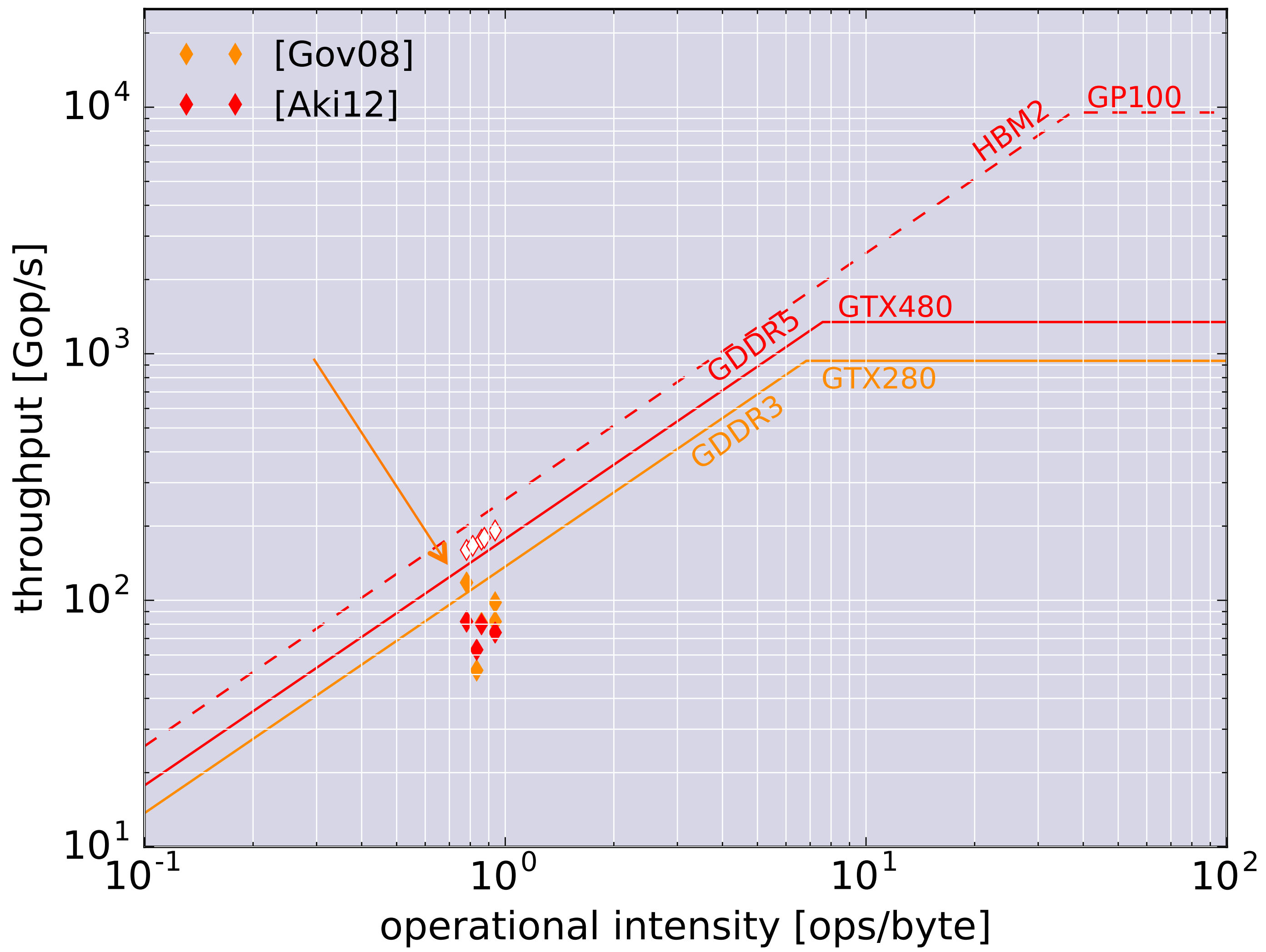
[Gov08]

FFT size:

- Small N ≤ 256 not enough threads.
- Medium 512 ≤ N ≤ 1024 data fits in on-chip shared memory
- Large 2048 ≤ N on-chip shared memory too small …

… and throughput is limited by DRAM bandwidth for each 1D-FFT radix-8 stage!

2D-FFT on GPUs



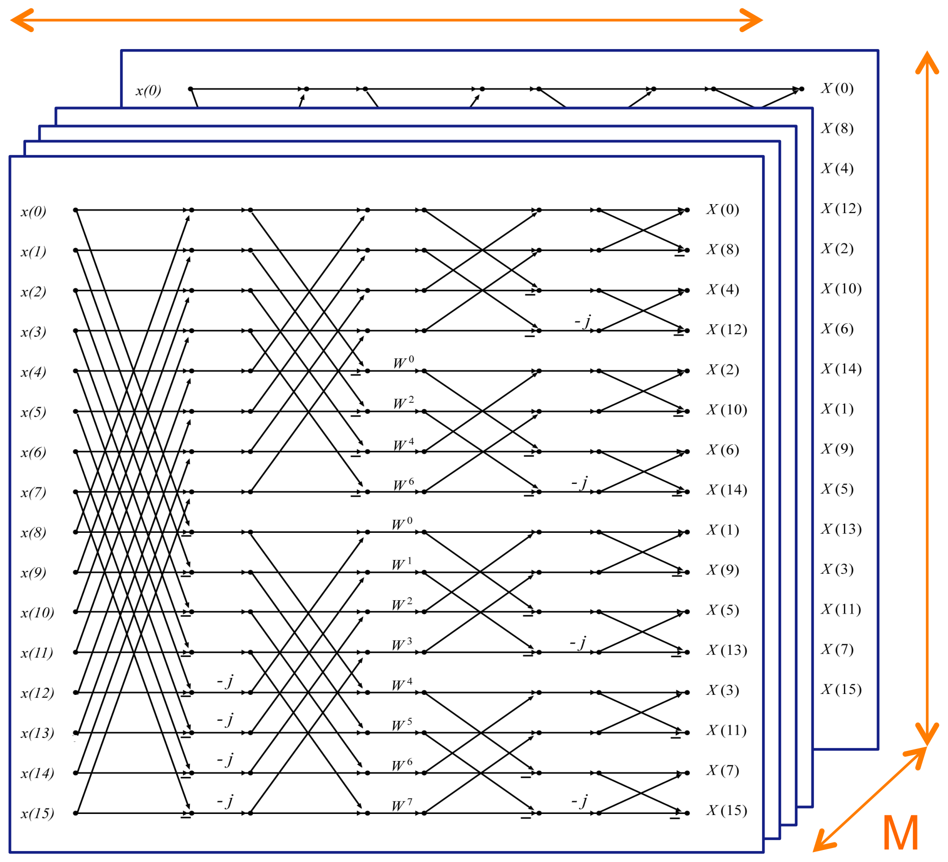
GP100:
throughputs based on I_{op}, 20% margin.

[Gov08]:
outlier for N=1024:
1D-FFT just fits in on-chip memory

Parallelism used for FFT on FPGAs vs GPUs

Multi-stage || (pipelined FFT):

- FPGA: simple and efficient;
- GPU: impractical (sync overhead, insufficient on-chip memory).



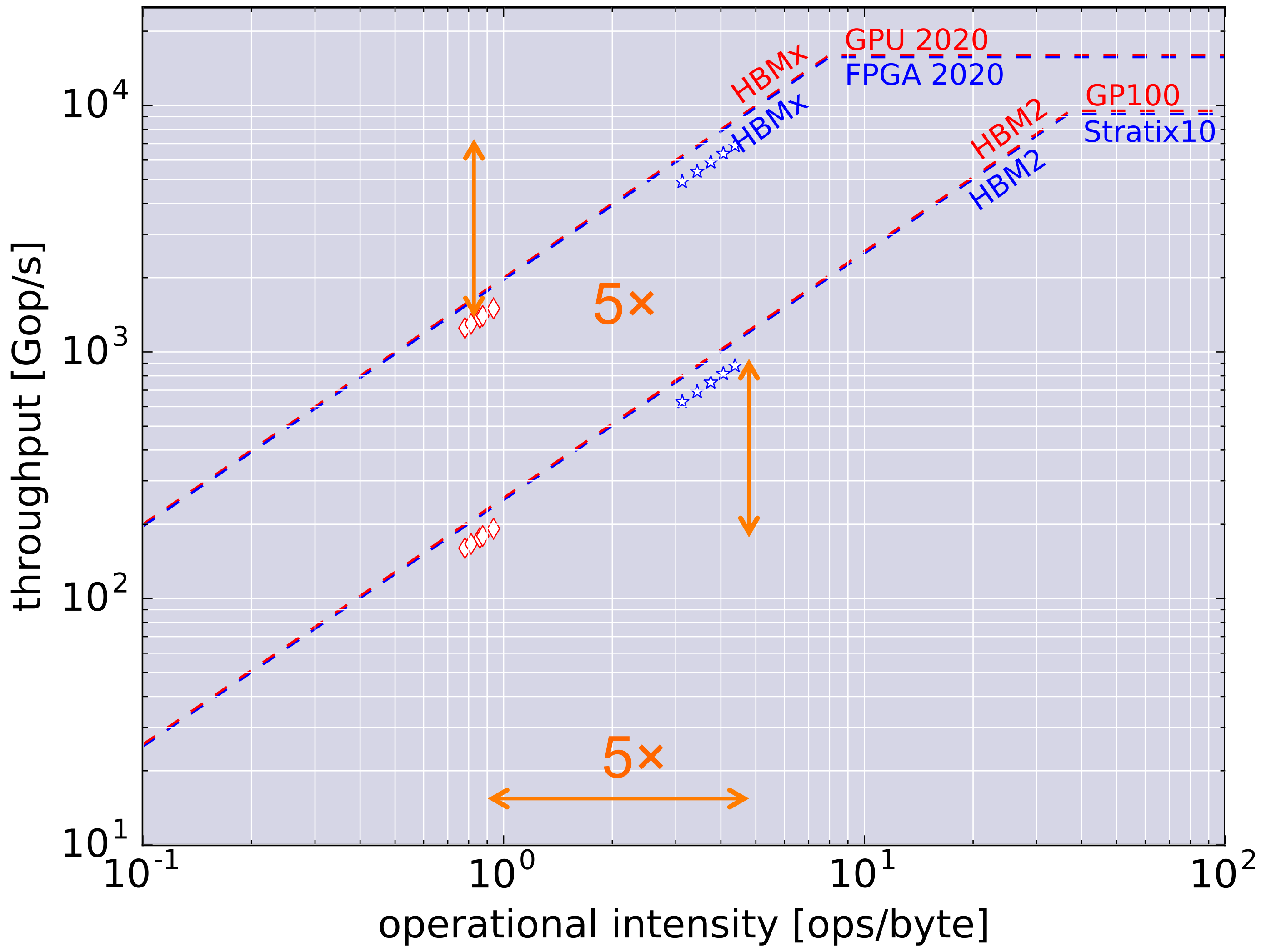
Intra-stage || (multi-butterfly):

- FPGA: not needed;
- GPU: essential to obtain sufficiently many threads.

Multiple FFT ||:

- FPGA: used to match throughput of M pipelines with memory bandwidth;
- GPU: needed to obtain sufficiently many threads.

Projected 2DFFT throughputs for GPU and FPGA



Y2020 GPU
numbers from
Nvidia paper [Ore14].

Y2020 FPGA
same “HBMx”;
similar mix of on-chip
resources assumed.

Large 2D-FFT: GPU or FPGA?

State-of-the-art FPGAs and GPUS: similar {GFLOP/s, GB/s, ridge points}

2D-FFT on FPGA: fairly good operational intensity (up to 5 op/byte):

- FPGAs support for pipelined 1D-FFTs and *B* (segmented) columns in ||.

2D-FFT on GPU: poor operational intensity (< 1 op/byte):

- requires many threads per scalar processor to hide pipeline and memory latencies; most die area is spent on register files;
- GPUs only support butterfly and multi-FFT parallelism.

For 2D-FFT, with *N* in the range 4k-16k, FPGAs relative to GPUs:

- require ≈ 5× less DRAM read-write passes,
- offer ≈ 5× more throughput,
- and require ≈ 10× less energy per 2D-FFT, …

… “on paper”.

FPGA as accelerator for exascale computing?

FPGA for radio astronomy (science data processing)?

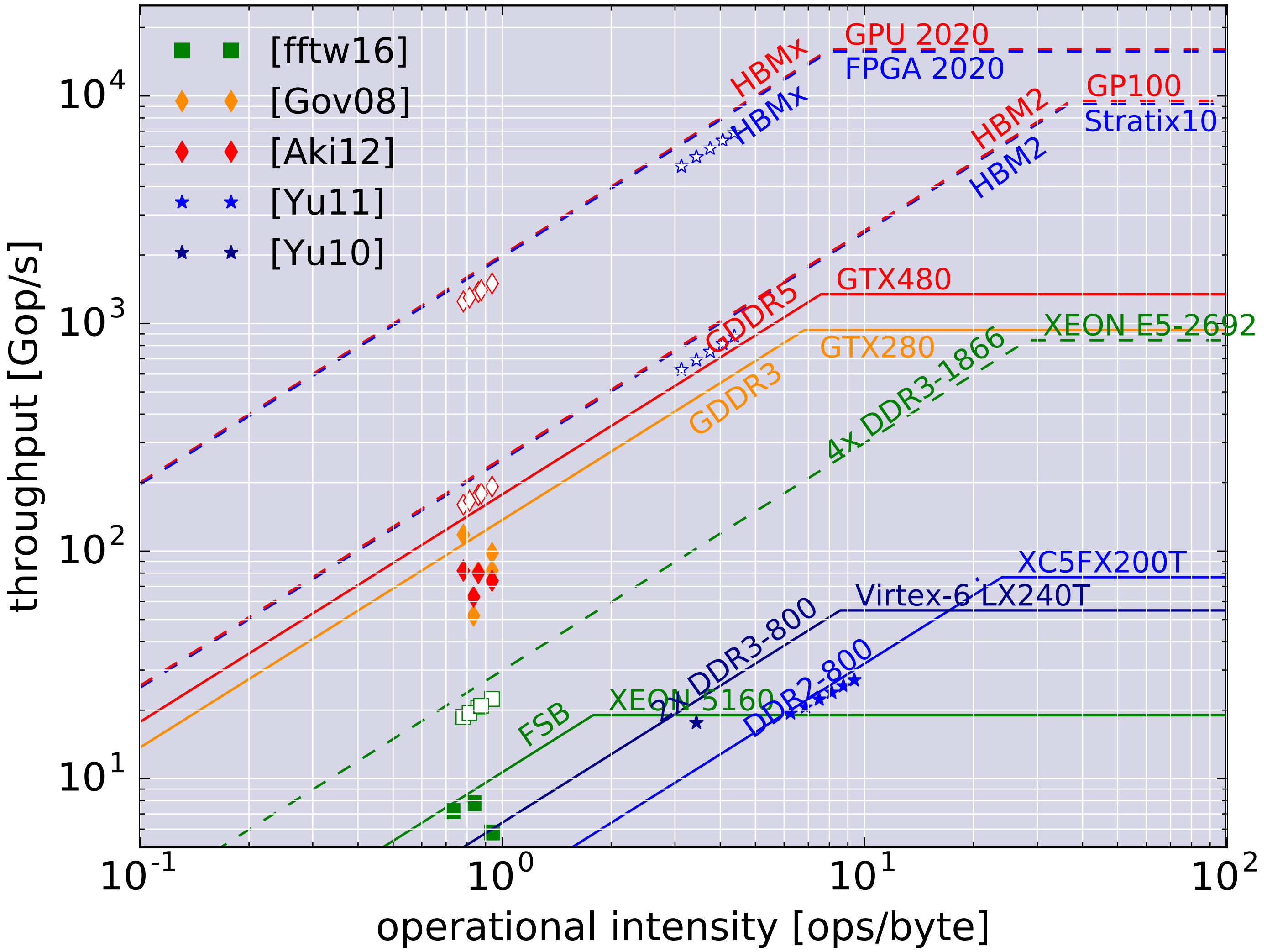
- “5× more throughput at 10× less power for 2D-FFT”
- … needs demo on HW,
- … and may just meet SKA power target (100 GFLOPs/s/W).
- How about other algorithms? gridding, w-projection, coherentdedispersion, …?

FPGA for exascale computing?

- Top 20 of top 500: 5× GPU (incl. #2 = Titan) versus 0× FPGA.
- “Intel + Altera = Efficient HPC Co-processing” (Altera website).
- Will “high-level programming model in OpenCL” deliver?

- FPGA for HPC momentum?

Several rooflines and 2D-FFT data points



Technische Universiteit
Eindhoven
University of Technology

References (1)

[Aki12] Berkin Akın et al, Memory Bandwidth Efficient Two-Dimensional Fast Fourier Transform Algorithm and Implementation for Large Problem Sizes, 2012 IEEE 20th Annual Int. Symp. on Field-Programmable Custom Computing Machines (FCCM), pp. 188 – 191.

[Bar13] R. F. Barret et al, On the Role of Co-design in High Performance Computing, *Transition of HPC Towards Exascale Computing, IOS Press, 2013, pp 141-155.*

[Cla90] B.G. Clark, Coherence in Radio Astronomy, pp. 1-10 in [Tay99].

[Dew13] P.E. Dewdney et al., SKA1 System Baseline Design, tech. report SKA-TEL-SKO-DD-001, SKA, Mar. 2013; www. skatelescope.org/?attachment id=5400.

[fftw16] http://www.fftw.org/speed/CoreDuo-3.0GHz-icc/

[Gov08] N.K. Govindaraju et al, High Performance Discrete Fourier Transforms on Graphics Processors, Proc. of the 2008 ACM/IEEE conference on Supercomputing, article No. 2.

[Gre14] The Green500 List - November 2014, http://www.green500.org.

[Hög74] Jan Högbom, Aperture Synthesis with a Non-Regular Distribution of Interferometer Baselines, Astronomy and Astrophysics Supplement, 19974Vol. 15, pp. 417-426.

[Jon14] R. Jongerius, S. Wijnholds, R. Nijboer, and H. Corporaal, “End-to-end compute model of the Square Kilometre Array,” *IEEE Computer*, Sept. 2014, pp. 48-54.

[Loa92] C. Van Loan, *Computational frameworks for the fast Fourier transform*. SIAM, 1992

[Ore14] Oreste Villa et al, Scaling the Power Wall: A Path to Exascale, SC14: Intl Conf. for High Performance Computing, Networking, Storage and Analysis, pp. 830-841.

References (2)

[Tay99] G.B. Taylor, C.L. Carilli, and R.A. Perly (eds.), Synthesis Imaging in Radio Astronomy II, ASP Conf Series, Vol. 180, 1999.

[Tho01] Thompson, A., Moran, J., & Swenson, G. 2001, Interferometry and synthesis in radio astronomy, Wiley, New York.

[Ver15] Erik Vermij et al, "Challenges in exascale radio astronomy: Can the SKA ride the techn-ology wave? Intl. Journal of High Performance Computing Applications 2015, Vol. 29(1), pp. 37-50.

[Wijn14] S. J. Wijnholds, A.-J. van der Veen, F. De Stefani, E. La Rosa, A. Farina, Signal Processing Challenges for Radio Astronomical Arrays, 2014 IEEE ICASSP, pp. 5382-86.

[Wil09] Samuel Williams, Roofline: an insightful visual performance model for multicore architectures, Comm. of the ACM, Volume 52 Issue 4, April 2009, pp. 65-76.

[Won10] H. Wong et al, Demystifying GPU microarchitecture through micro-benchmarking, 2010 IEEE Intel. Symp. on Performance Analysis of Systems & Software (ISPASS), pp. 235 – 246.

[Yu10] Chi-Li Yu et al, Bandwidth-intensive FPGA architecture for multi-dimensional DFT, 2010 IEEE Intl. Conf. on Acoustics, Speech and Signal Processing, pp. 1486 – 1489.

[Yu11] Chi-Li Yu et al, FPGA Architecture for 2D Discrete Fourier Transform Based on 2D Decomposition for Large-sized Data, Journal of Signal Processing Systems, July 2011, Volume 64, Issue 1, pp. 109-122.