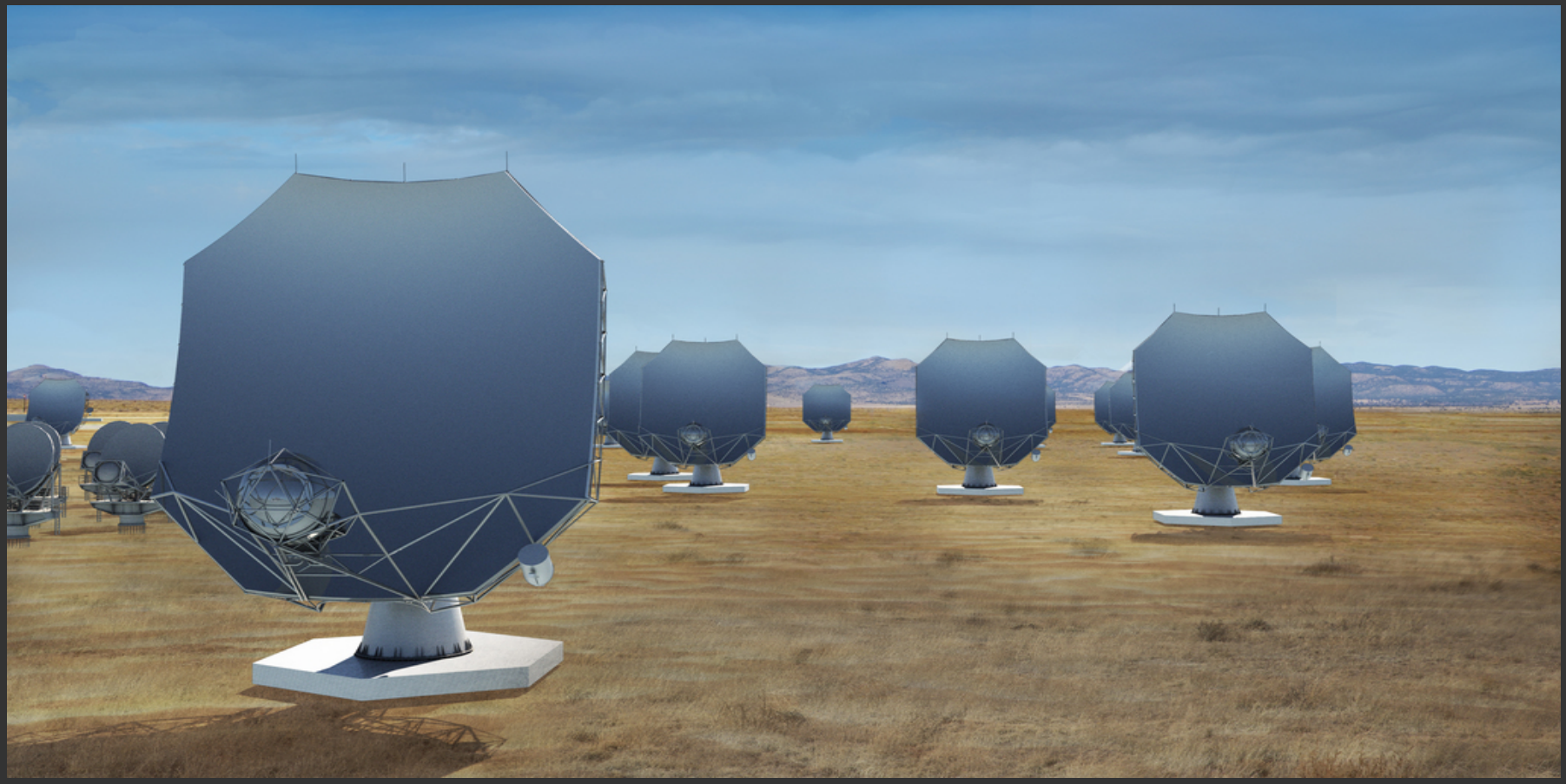
Algorithms R&D and ngData Processing



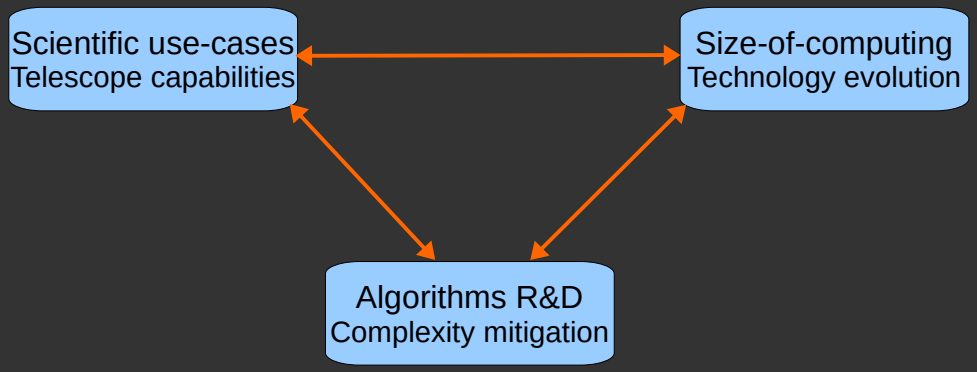
Sanjay Bhatnagar
Algorithms R&D Group, NRAO
July 31st 2023

Goal

- Develop strategic thinking:

  Understand telescope capabilities, science requirements and technology predictions in the coming decade to develop scalable algorithms/architecture in the context of the ngVLA and ALMA upgrade.

- Algorithm R&D: Navigate complex interaction between three drivers

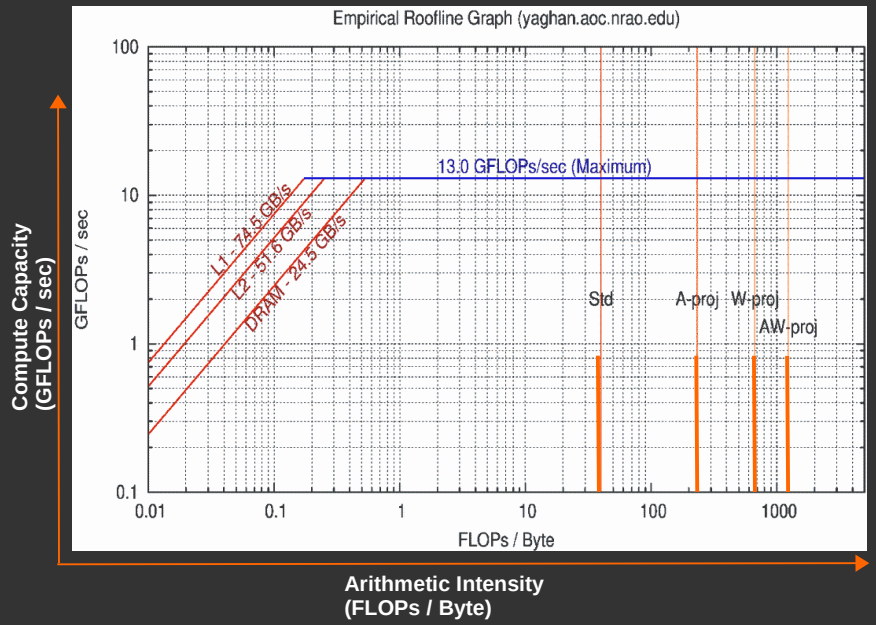


- The process we followed:
  - Derive required algorithms → size-of-computing
  - Develop scalable algorithm- and computing-architecture
  - Prototype to test performance, scaling and flexibility

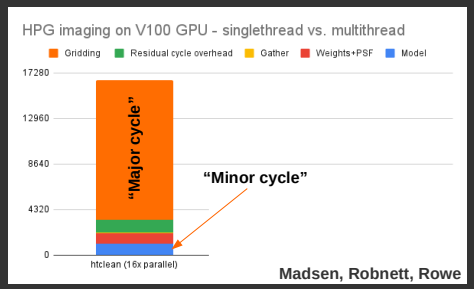
ngVLA Size of computing estimates

- Establish the algorithms *required* for the key science goals (KSGs)
- Develop a theoretical model for the computational complexity
  - Develop scaling laws
  - Identify the highest-nail(s) in an end-to-end data processing pipeline

- Measure the code efficiency on a *single-core* of *an* implementation.
  - Compare against industry standard to normalize the s/w and h/w used.
- Estimate and verify the SofC based on code efficiency.



- **Compute bound on CPUs**
- **Memory I/O bound on GPUs**



Run-time stack for Imaging

ngVLA Computing Memo #4

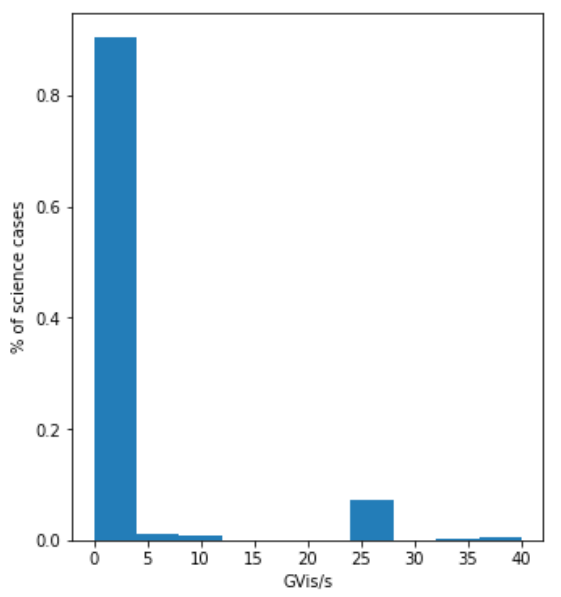
Size of computing

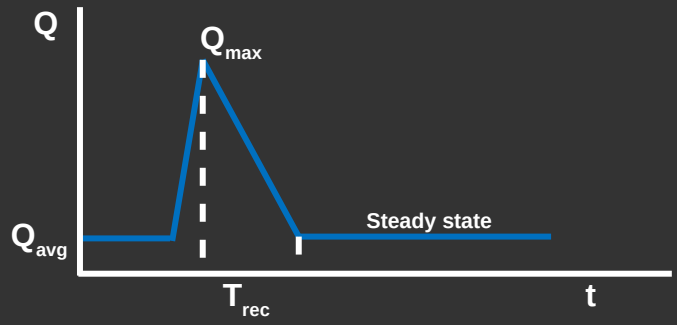
- ngVLA
  - Average over all KSGs: 50 PFLOP/s. <10% of use-cases require ExaFLOP/s
    - » Data rates weighted by the fraction of telescope time.
  - SKA: Latest estimates are similar

[ngVLA Computing Memo #4]

- Used Queuing Theory to model the trade-off between latency and affordability: Peak vs steady-state queues
  - 50 PFLOP/s corresponds to T_{rec} = O(1 day)

[Heriart, in prep]

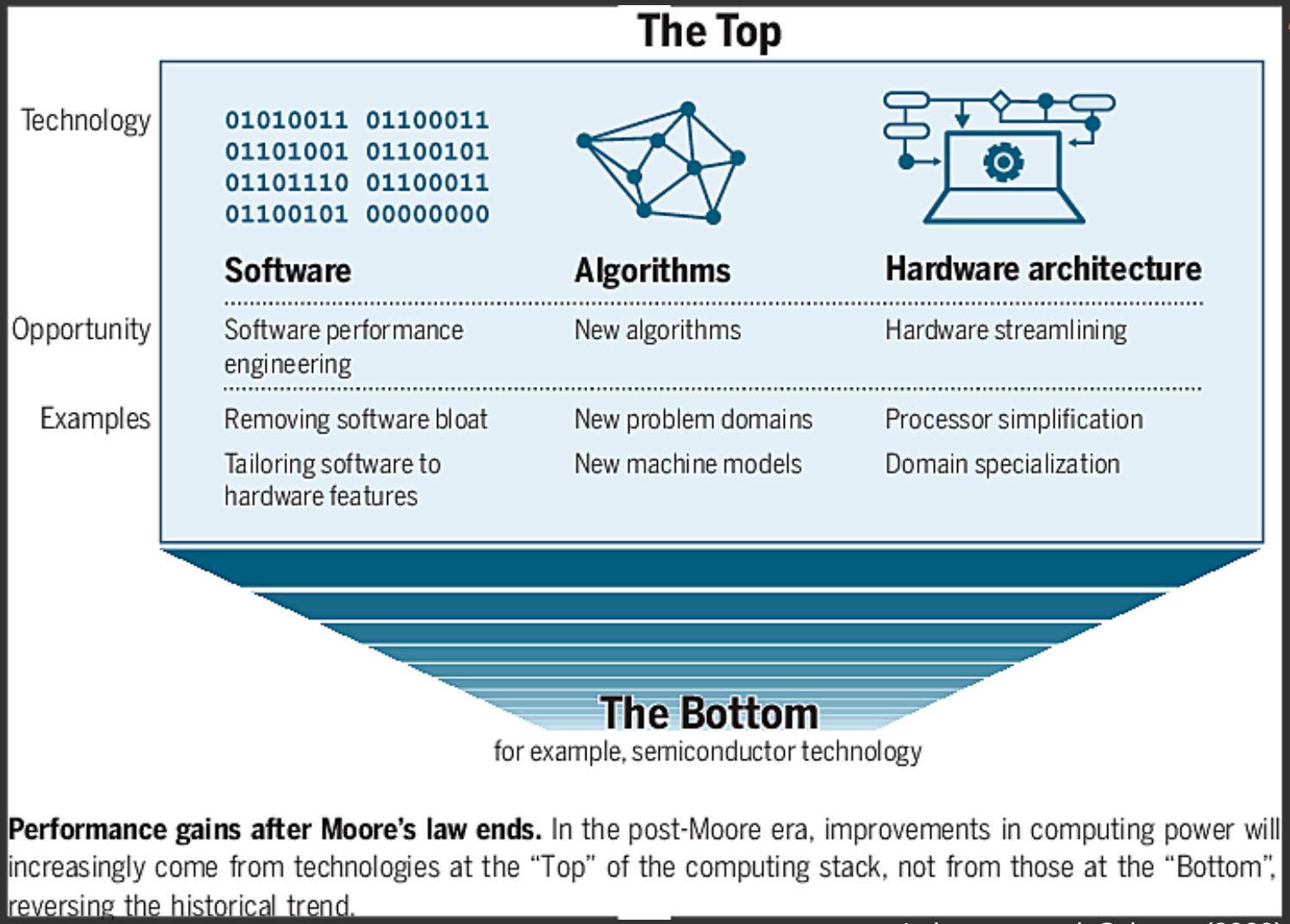




Challenges

- The estimated *size of the computer* for the ngVLA
  - Need O(Million)-way parallelization with CPU cores running 24x7!
  - Some of the largest *facilities* are ~40 PFLOP/sec machines (peak!)
- Large-scale parallelization is essential – even with technology predictions
- Total available computing capacity:
  - A typical desktop: 0.1 TFLOP/s
  - A typical GPU: 10 TFLOP/s
  - However, best-case usage efficiency is in single-digit percentages
- How do we harvest the available capacity?
  - Parallelization at multiple scales
  - New kind of h/w + Algorithms and software to efficiently utilize it (more on this later…).
  - A flexible architecture that can adopt to rapid tool-chain evolution.
  - Managing the resulting level of complexity.

Future outlook: The Computing Stack



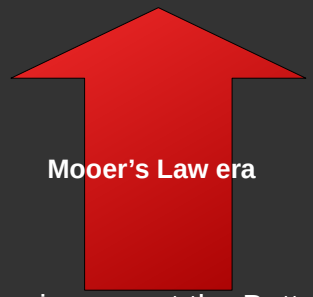
Performance gains after Moore's law ends. In the post-Moore era, improvements in computing power will increasingly come from technologies at the "Top" of the computing stack, not from those at the "Bottom", reversing the historical trend.

Leiserson et al. Science (2020)

"There's plenty of room at the Top"
- Leiserson et al. (2020)

Current capacity utilization:
In single-digit percentage

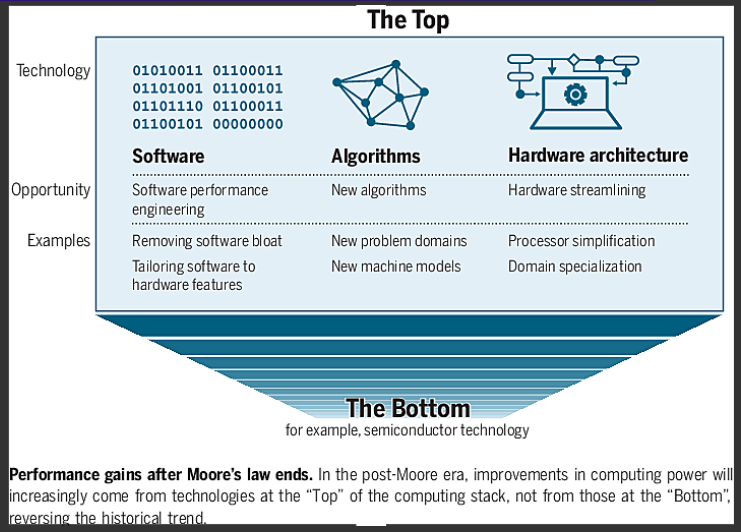
Need algorithms with high Arithmetic Intensity



"There is room at the Bottom"
- Feynman (1959)

The Computing Stack: The Bottom & The Top

- The Bottom: Moore's-Law era
  - Runtime reduced by 2x if one just waited
  - Improvements were more predictable
  - Led to Sys-on-Chip → Multiple cores
    - Not all of the h/w is used by any given application
    - Not cost-effective in construction cost (silicon yield), or operating costs (power).
    - Not all algorithms are a good fit: I/O, Memory bottlenecks

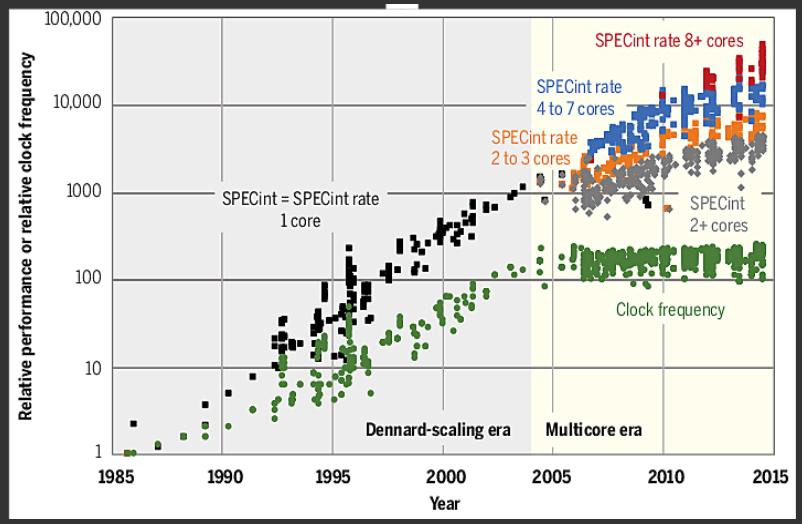


Performance gains after Moore's law ends. In the post-Moore era, improvements in computing power will increasingly come from technologies at the "Top" of the computing stack, not from those at the "Bottom", reversing the historical trend.

- Number of GP cores now is also limited by the end of Moore's-law era.

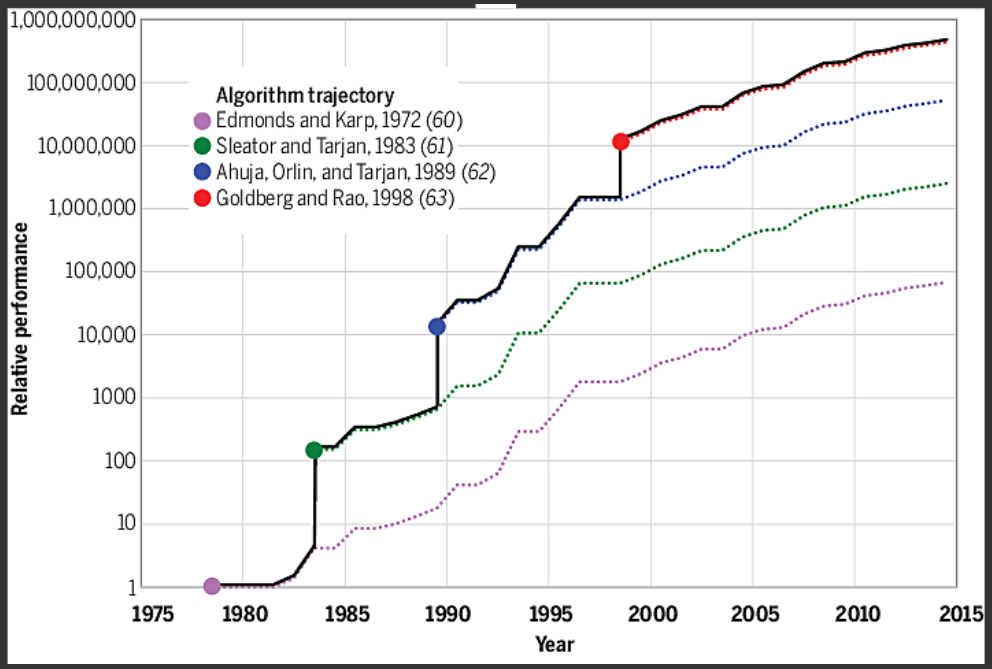
- The Top: Post Moore's-Law era:

  Massively parallel h/w of simpler cores (not GP)
  - Improvements will come from performance engineering, new algorithms, better silicon utilization
  - Algorithms that effectively parallelize on multiple scales of the problem



The Computing Stack: The Algorithms

- Algorithms R&D
  - Historically AR&D has delivered runtime gains comparable to the Moore's Law
  - Moore's Law has historically caught up...but that has now ended!



- Progress is more sporadic compared to the Moore's law

- Outlook:
  - Algorithms with higher Computational Intensity (Compute-to-I/O ratio) + cache-friendly
  - Data locality will matter more
  - General-purpose vs bespoke software (and hardware!)

Scalable Algorithm Architecture

- Architecture must be flexible for the evolving computing needs, algorithms, computing h/w and s/w
- A mathematical framework based on fundamental Physics/Optics, Sig. Proc.

https://www.aoc.nrao.edu/~sbhatnag/misc/AlgoArch.pdf

$$V^{obs} = \boldsymbol{G}^{\boldsymbol{M}} \boldsymbol{F} B^{M} I^{M} + noise \qquad \chi^2 = \sum_i \left| Data_i - Model_i(P) \right|^2$$

$$P_i^{k+1} = P_i^k + [H_{ij}]^{-1} f\left(\frac{\partial \chi^2}{\partial P_i^k}\right) \quad ; \qquad [H_{ij}] = \frac{\partial^2 \chi^2}{\partial P_i^k \partial P_j^k}$$

Model update | Step size | Derivative

- Self Cal: $g_i^{k+1} = g_i^k + \alpha f\left(\frac{\partial \chi^2}{\partial g_i^k}\right)$
- Pointing SelfCal: $p_i^{k+1} = p_i^k + \alpha f\left(\frac{\partial \chi^2}{\partial A}\frac{\partial A}{\partial p_i^k}\right)$
- Imaging $I^{M^{k+1}} = I^{M^k} + \alpha f\left(\frac{\partial \chi^2}{\partial I^{M^k}}\right) \quad ; \quad \frac{\partial \chi^2}{\partial I^{M^k}} = Residual\ Image$

Algorithm Architecture: Components view

- Mathematical framework is the same for calibration and imaging
- Specialization of the components delivers various calibration and imaging algorithms

$$P_j^{k+1} = P_j^k + [H]^{-1} f\left(\frac{\partial \chi^2}{\partial P_j^k}\right)$$

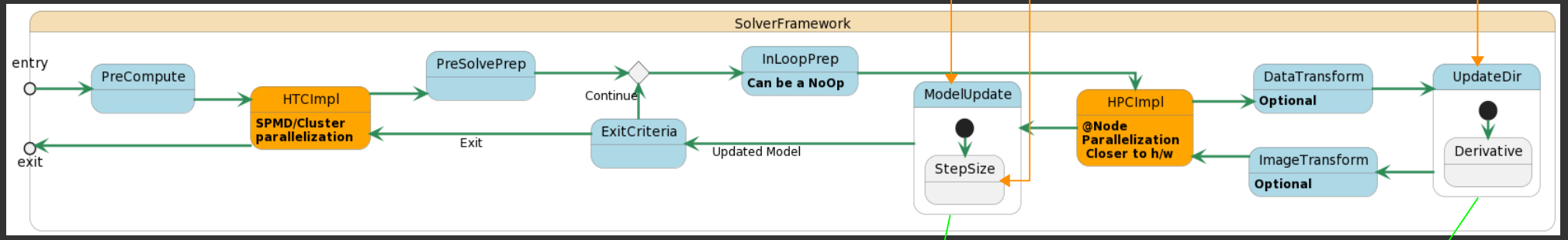
a.k.a. the "Minor cycle"

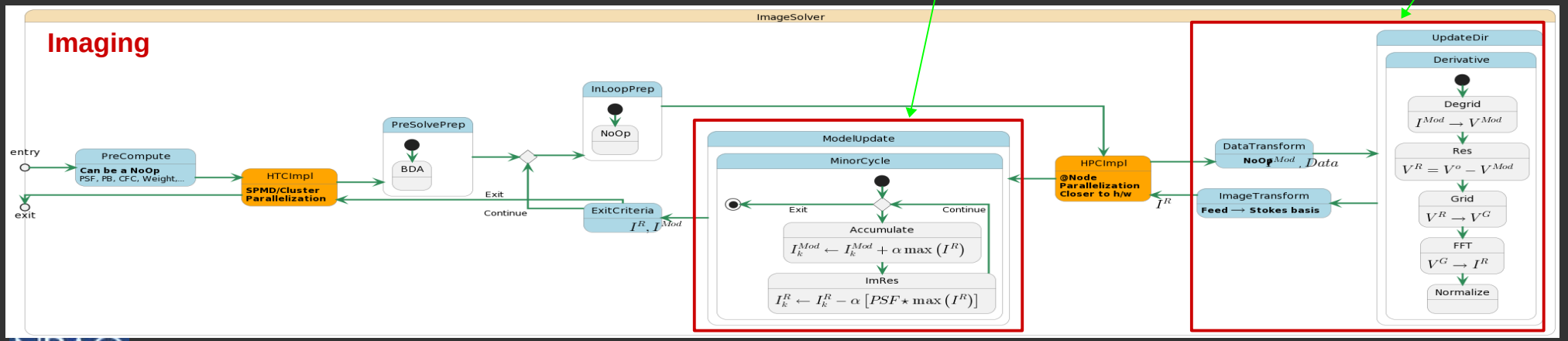
Update Model Image

a.k.a. the "Major Cycle"

Compute Residual Image

a.k.a. the "Loop gain"





Algorithm Architecture: Components view

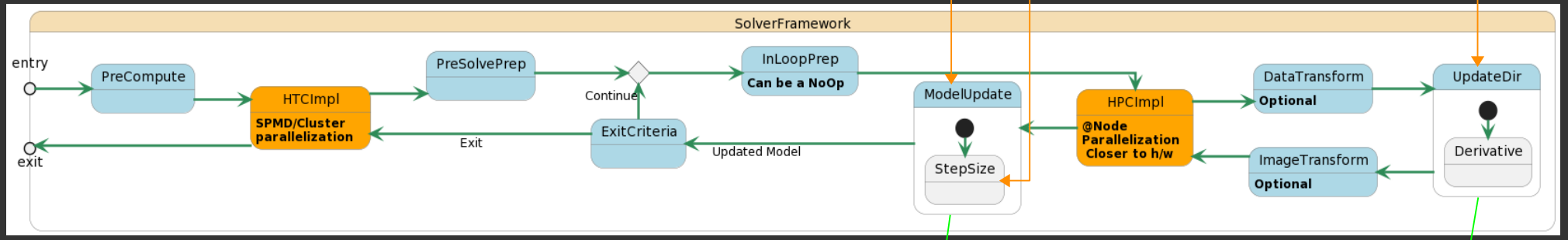
- Mathematical framework is the same for calibration and imaging
- Specialization of the components delivers various calibration and imaging algorithms

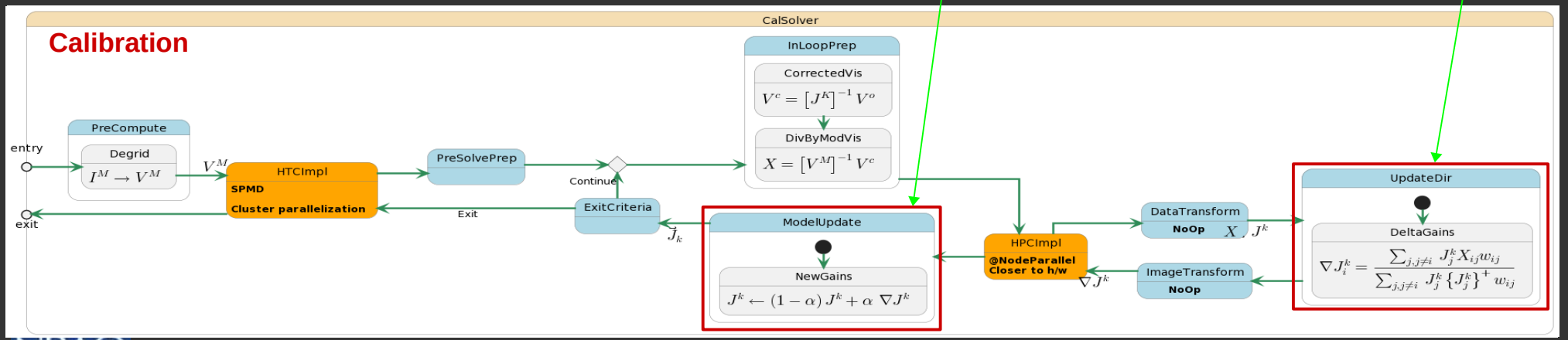
$$P_j^{k+1} = P_j^k + [H]^{-1} f\left(\frac{\partial \chi^2}{\partial P_j^k}\right)$$

Update gains

a.k.a. the "Loop gain"

Compute Derivative vector

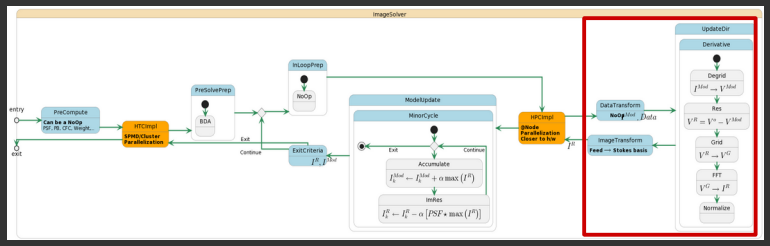




Calibration of DD Effects: Example of UpdateDir

- Measurement Equation

$$V_{ij}^{Cal}(\nu) = A_{ij}(\nu, t) * V^{True}(\nu, t)$$



- Find an operator X which when applied to the above equation, projects-out the undesirable effects of A

- Then $F \; X_{ij} \; V_{ij}^{Cal} = F \; V^{True} = I^{Raw}$

X encodes the Physics of the problem

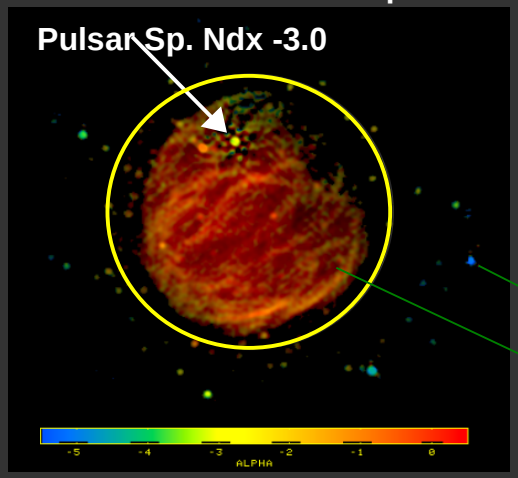
$$F\,X[V^{Obs} - X^{-1}F^{-1}I^{M}] = I^{Res} = \frac{\partial \chi^2}{\partial I^{M}}$$

Develop algorithms with high Arithmetic Intensity (trade off between i/o and FLOPs)

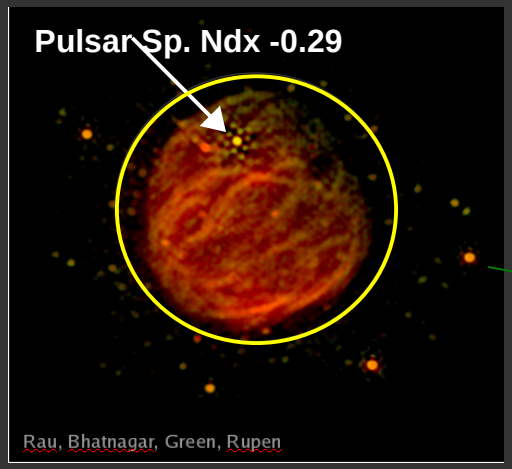
Wide-band AW-Projection

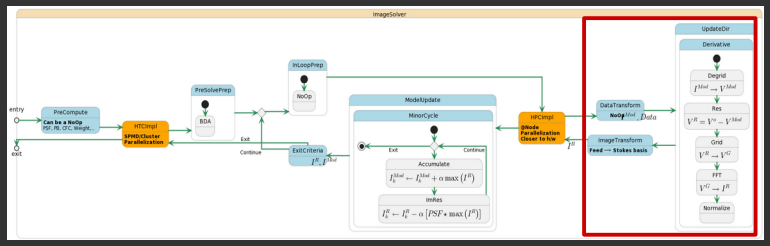
Effects of wide-band optics

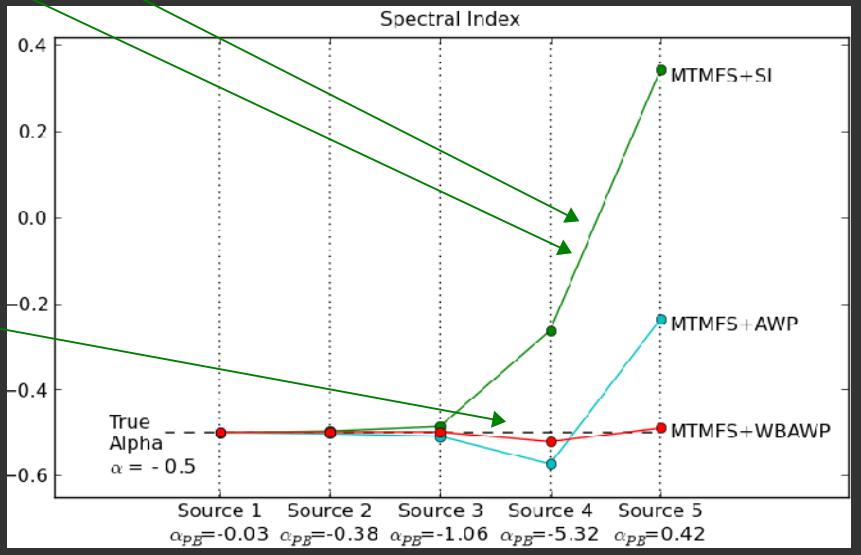
Standard imaging



WB A-Projection

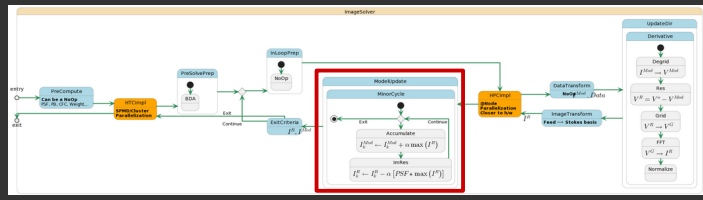






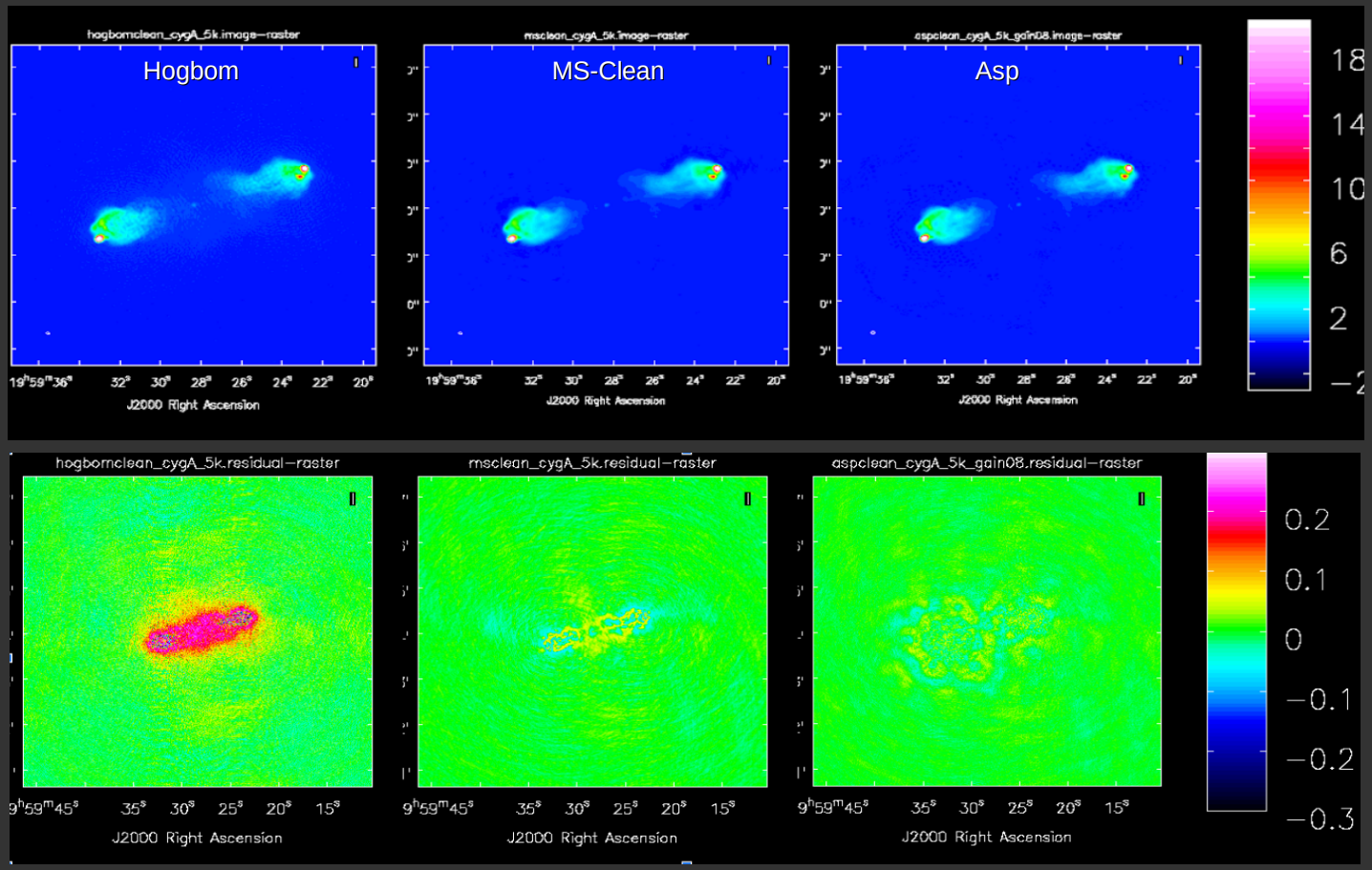
The Asp algorithm: Example of ModelUpdate

- Adaptive Scale Pixel (Asp): Scale-sensitive image reconstruction of complex emission
  - Asp-Clean: Narrow-band implementation
  - WAsp: Wide-band Asp



Stokes-I

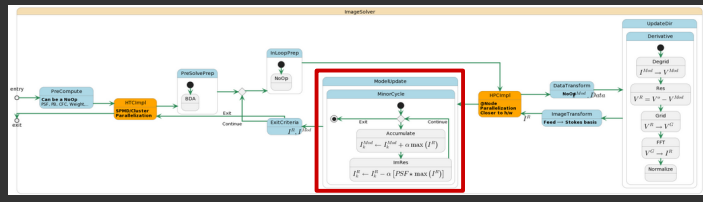
Residual



Hsieh et al. (in prep)

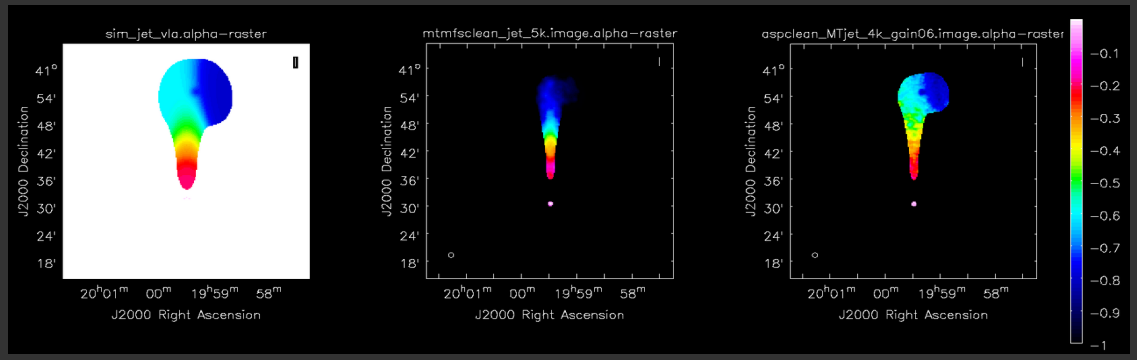
The WAsp algorithm

- Adaptive Scale Pixel (Asp): Scale-sensitive image reconstruction of complex emission
  - WAsp: Wide-band Asp



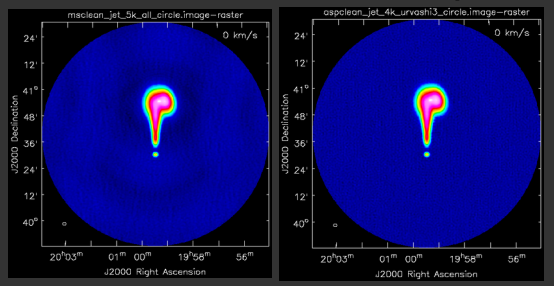
MS-MT-MFS **WAsp**

Spectral Index Mapping

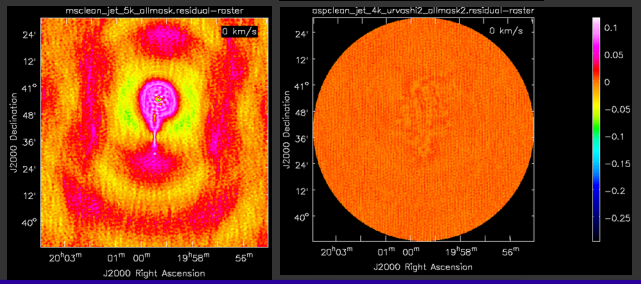


MS-MT-MFS **WAsp**

Stokes-I



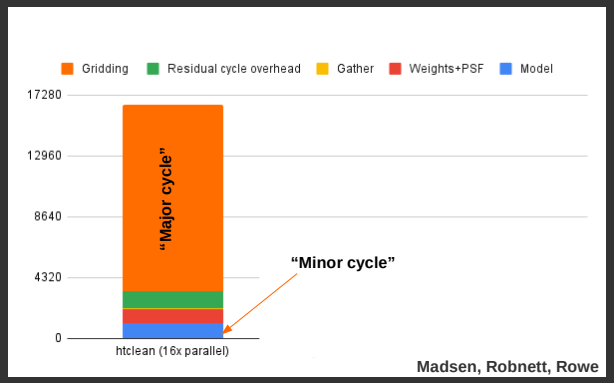
Residual



Hsieh et al. (in prep)

Arithmetic Intensity

- RA algorithms have high Arithmetic Intensity (Compute-to-i/o ratio)
  - Compute scaling: $N^2_{support} \times N_{vis}$: Dominated by residual image (the “Major cycle”)
  - Memory footprint: $N^2_{Scales} + N^2_{Terms}$: Dominated by Deconvolution (the “Minor Cycle”)



- Image reconsturction accounts for >90% of the computing cost in a “typical” end-to-end processing

- Heterogeneous h/w for optimal execution of the processing graph:
  - Cluster of cheaper CPU-GPU for the Major cycle
  - Fewer faster computers for the Minor Cycle

Algo Arch: Deployment on heterogeneous h/w

- Different components deployed on a variety of h/w resources
  - H/W scaling: CPU single/multi-cores, cluster, variety of GPUs,..., wide-area network of GPUs

    Algo scaling: Single pointing, pointed mosaic, pointing correction + MSCLEAN, Asp,...

Parallelization framework

Bash-, Py-scripts
HTCondor

Loosely connected
Network of computers

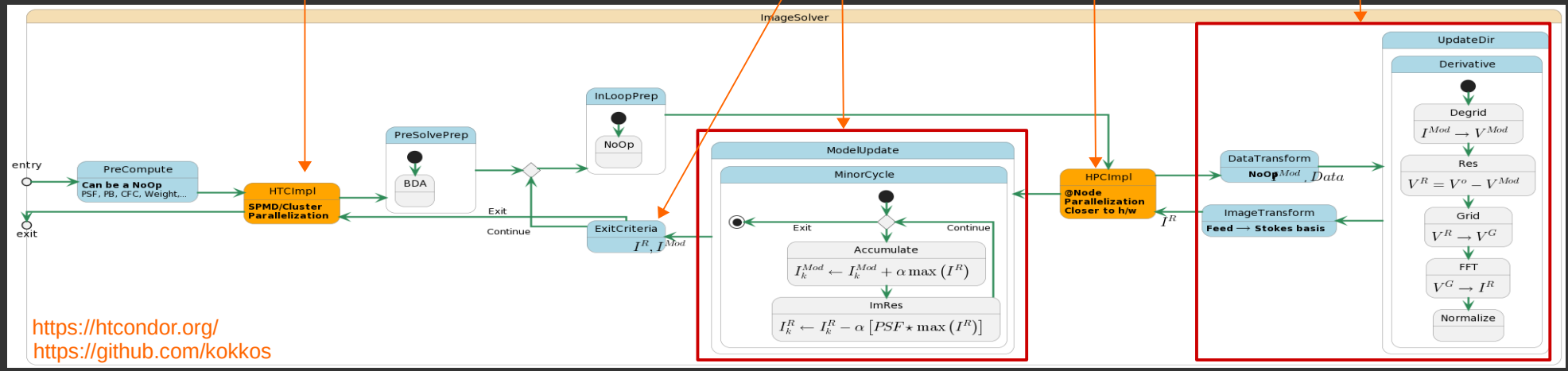
Deconvolver app

On a fast multi-core CPU

Multi-threaded CPU code
On the GPU-Host

Derivative app
Uses Perf. Engg. Tools (Kokkos)

On a single node (CPU or GPU)
On a Cluster of CPUs/GPUs
On a wide-area network of CPUsGPUs

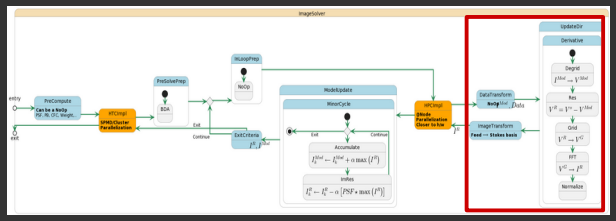


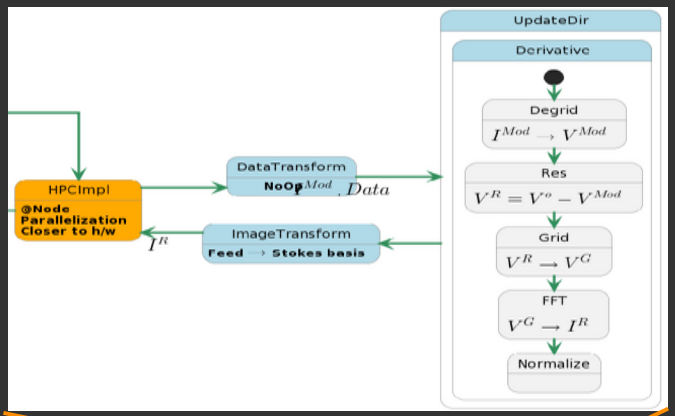
Resource hog: Needs large-n-fast memory
Large memory footprint

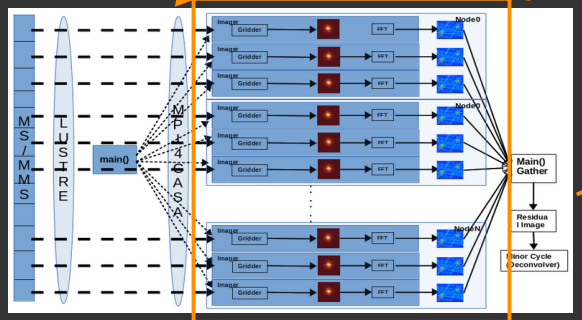
Resource hog: Needs compute cores
- High compute load
- Relatively lower memory footprint

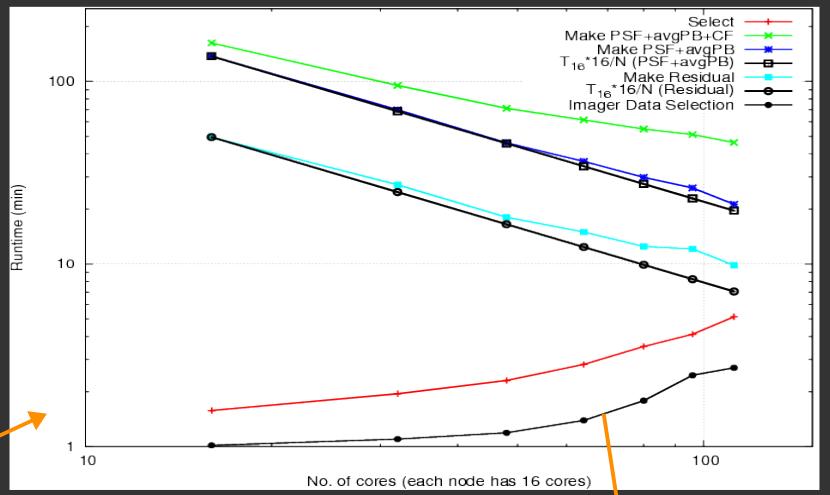
Scaling: On multi-CPU/cores hardware

- High Computational Intensity (FLOP per byte)
  - $O(10^{2-3})$ FLOP per data point. Number of data points: $O(10^{12-15})$







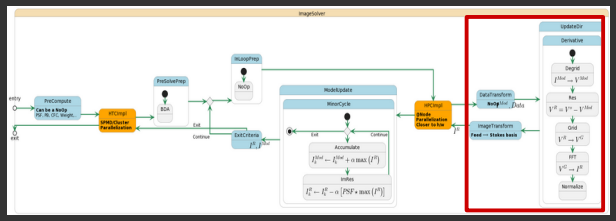


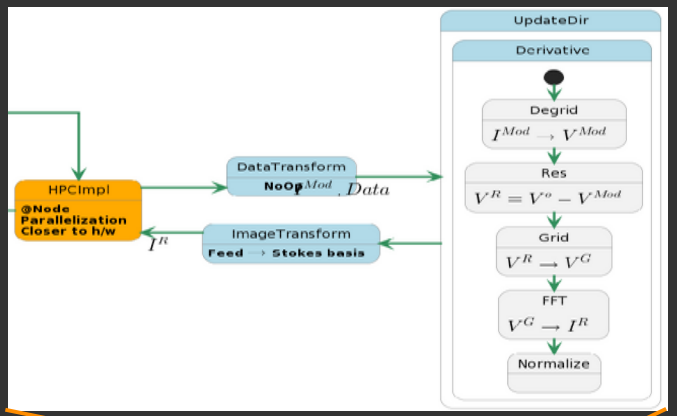
- Data scatter overheads, due to
  1. Partitioning monolith MS
  2. Organiztion of the bulk-data on disk
- Work in progress in CASA/ngCASA
  1. Differently organized on the disk
  2. Improvements in storage model

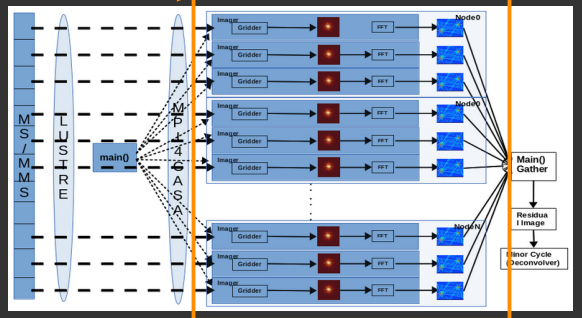
ngVLA would need O(Million)-way parallization!

Scaling: On massively parallel h/w

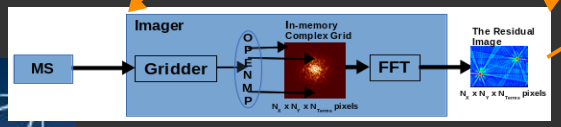
- High Computational Intensity (FLOP per byte)
  - $O(10^{2-3})$ FLOP per data point. Number of data points: $O(10^{12-15})$

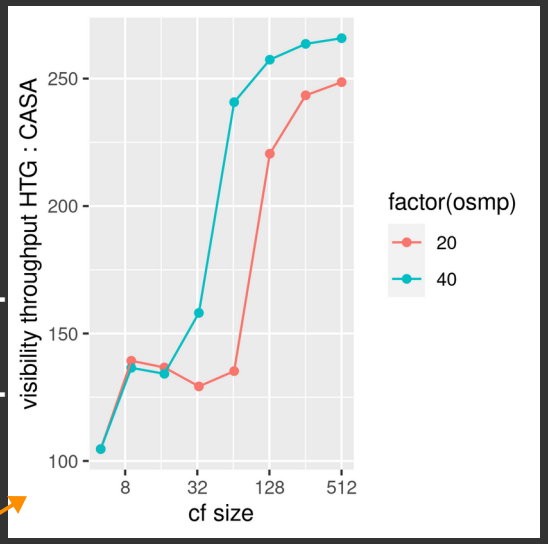




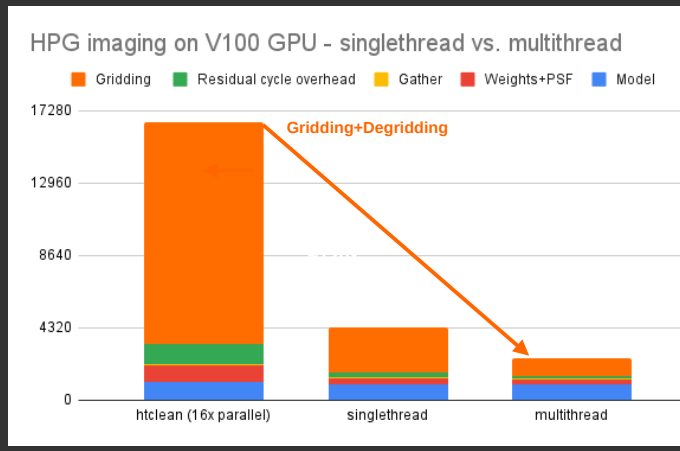


Complexity reduction





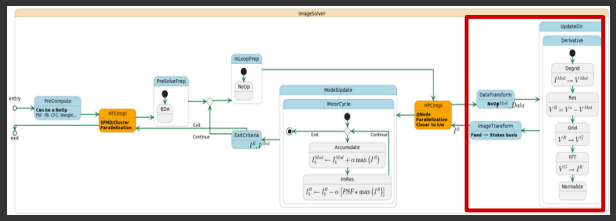
ngVLA Computing Memo #5, #7

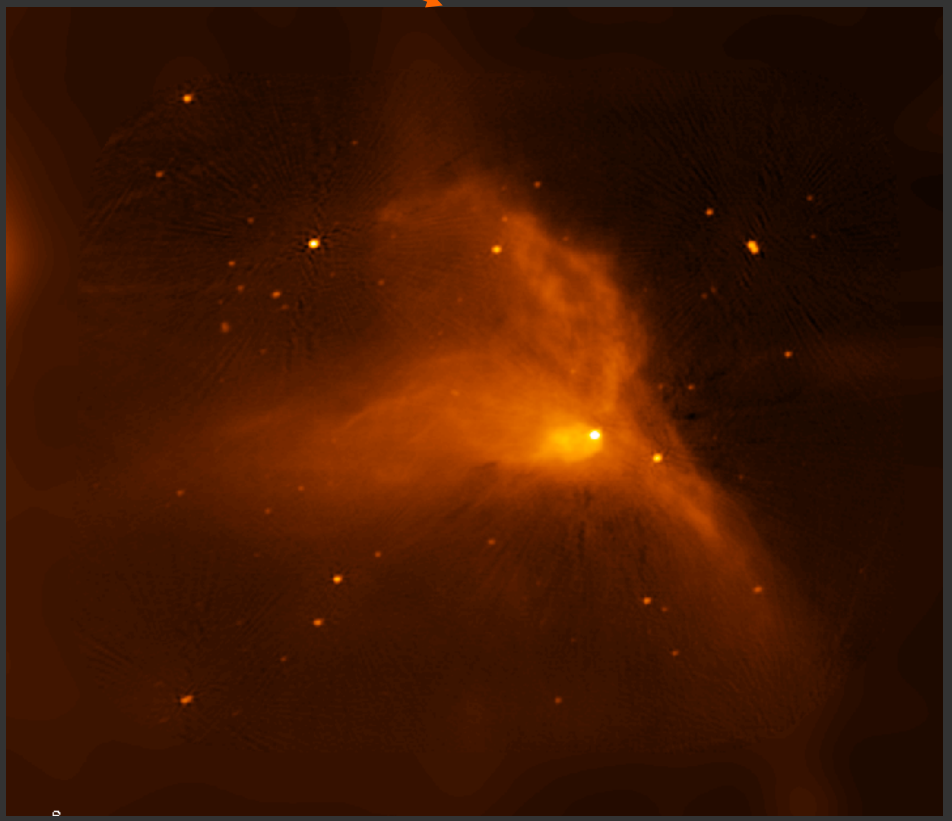


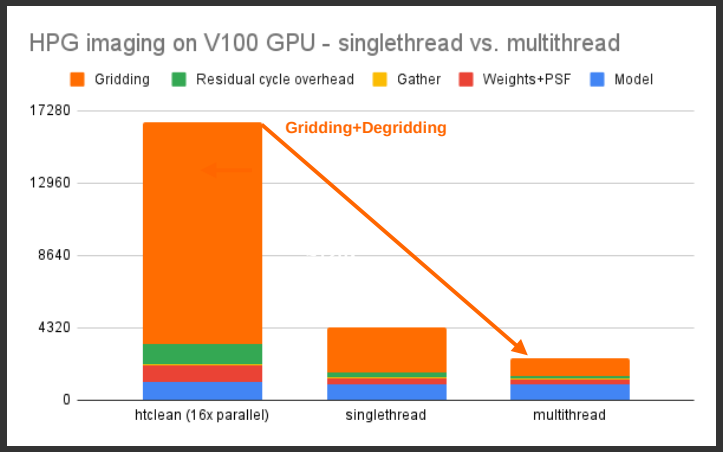
ngVLA would need O(10³)-way parallization!

The High Performance Gridder

- A gridder on a GPU connected to a CPU host (NGVLA Memo #05)
- What does it mean in real-life application?
  - 200-pointing wide-band mosaic: 7-10 days vs 2.5hr







Multi-core CPU

Single CPU-core +GPU

Multi CPU-cores +GPU

ngVLA would need O(10^3)-way parallization!

Scaling: On Wide-area network

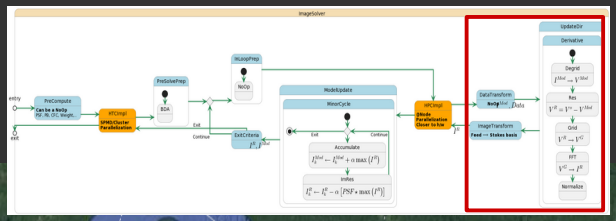
- High Throughput Computing:

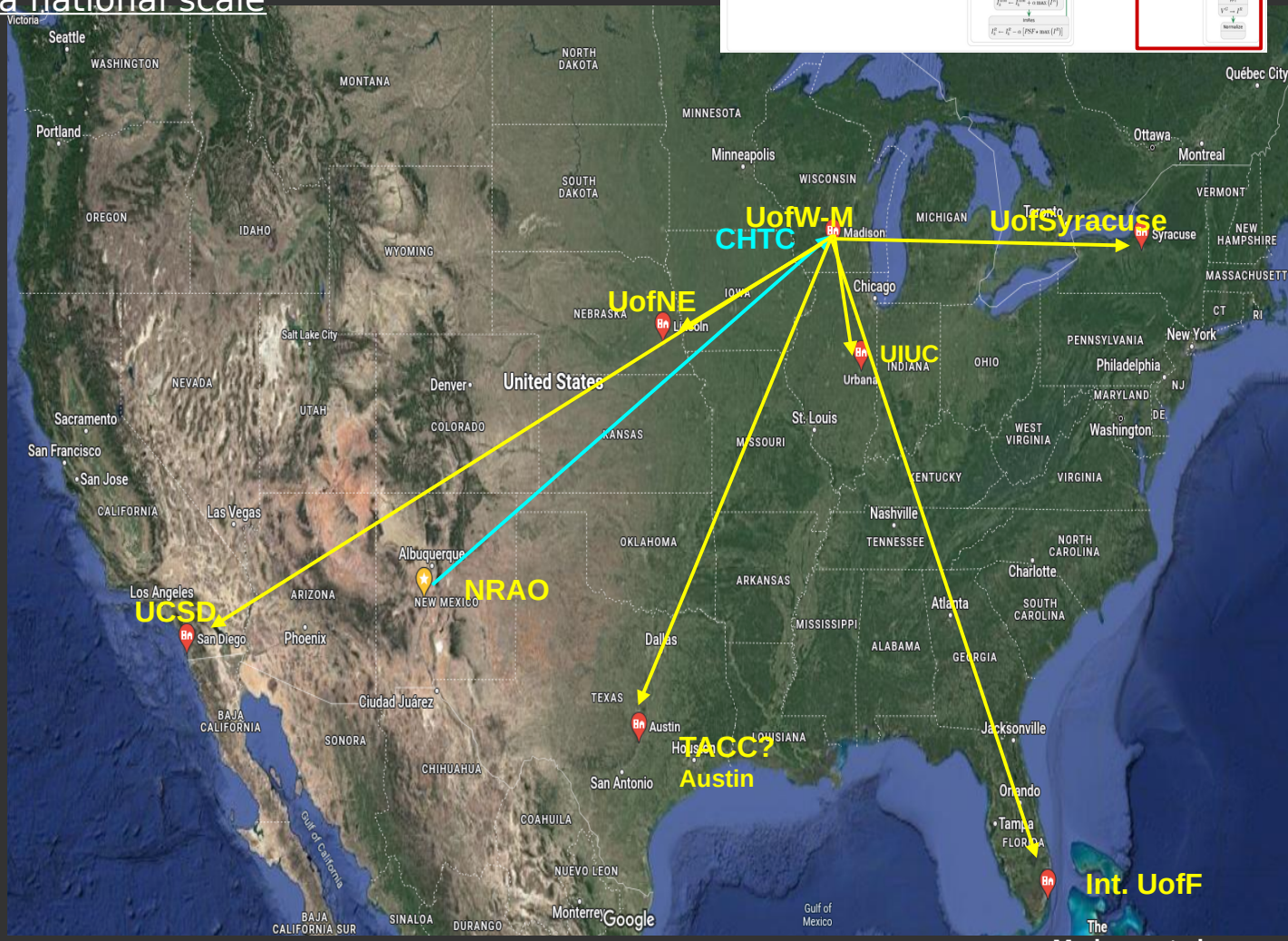
  Center for High Throughput Computing, U of W-M.

  GPU cluster: Computing at a national scale

- Trigger from NRAO, deploy nationally
- Opportunistic computing + Edge-caching

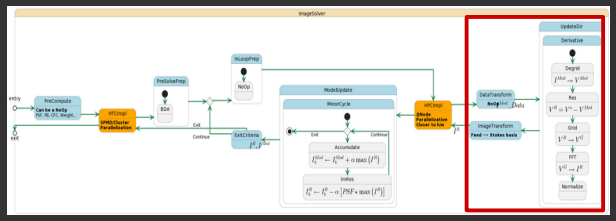
- Work in progress
  - More human and computing resources
  - International resources



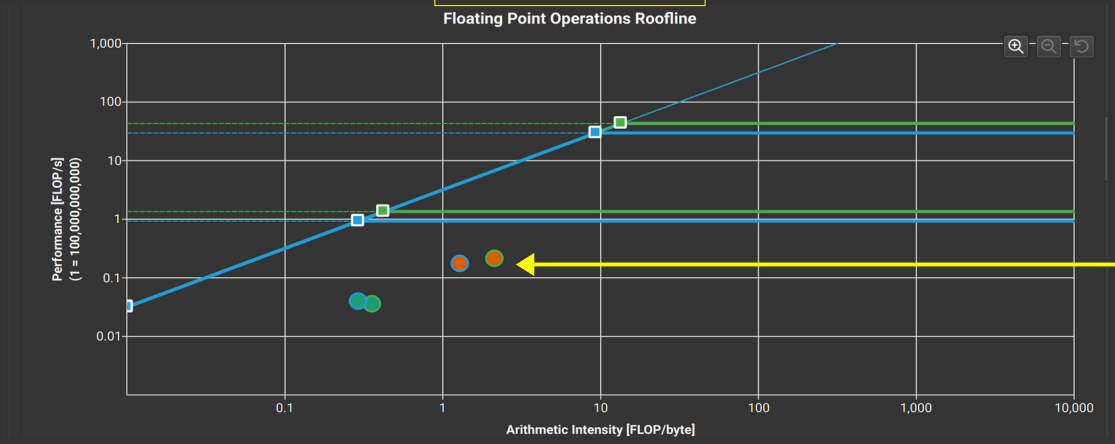


Scaling: On hardware generations

- Scaling on the GPU: View from the “inside”



NVIDIA T4 GPU



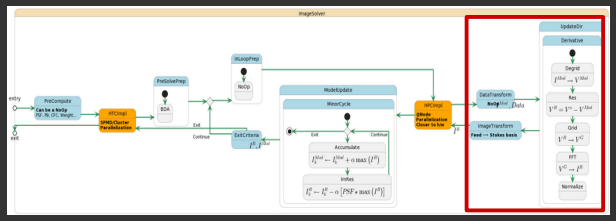
HPG is compute bound

[Courtesy Jagannathan, Heriart]

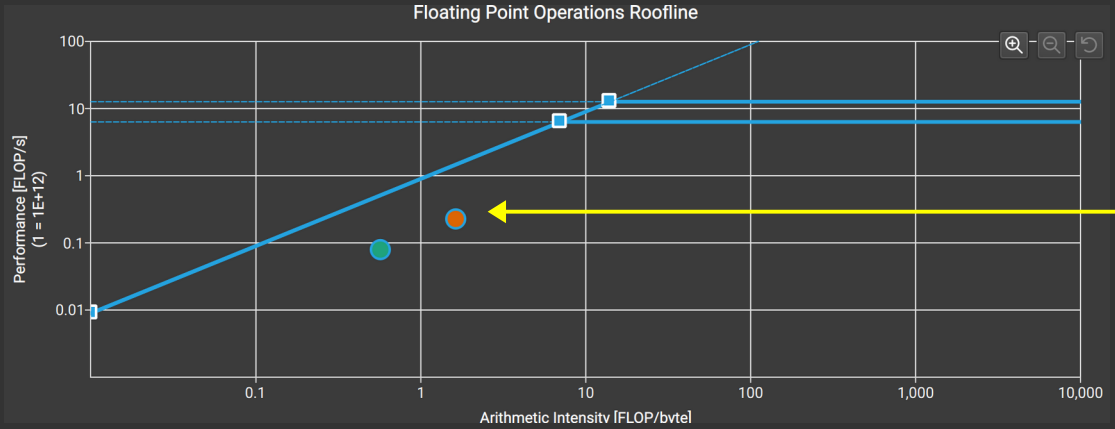
- Algorithms and implementation needs to be compute-limited for run-time to scale.
- Number of CUDA cores, FLOP rate is increasing with GPU generations. Memory bandwidth is not.

Scaling: On hardware generations

- Scaling on the GPU: View from the “inside”



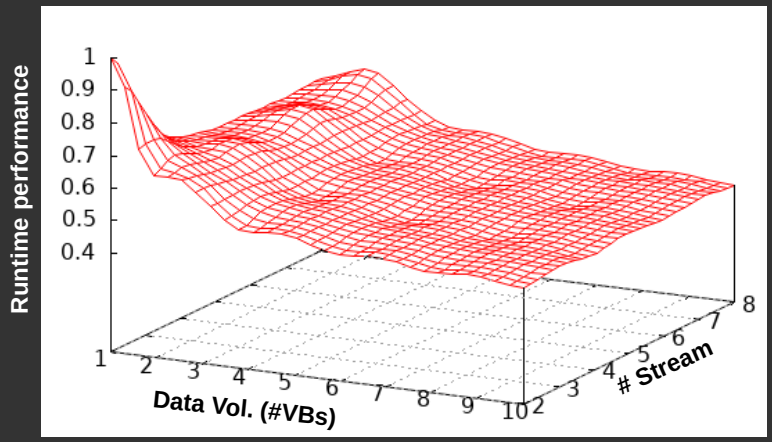
NVIDIA V100 GPU



HPG is memory-bandwidth bound

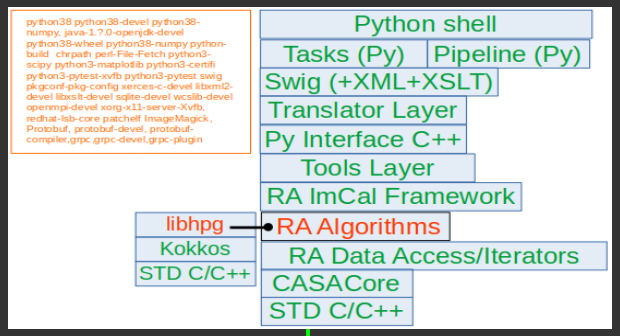
[Courtesy Jagannathan, Heriart]

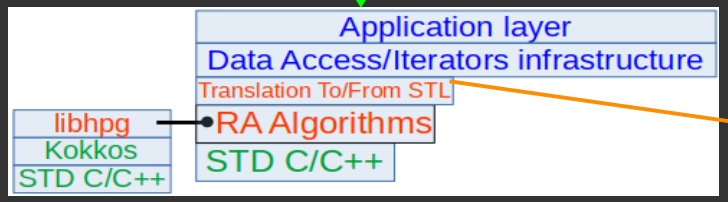
- Algorithms and implementation needs to be compute-limited for run-time to scale.
- Number of CUDA cores, FLOP rate is increasing with GPU generations. Memory bandwidth is not.

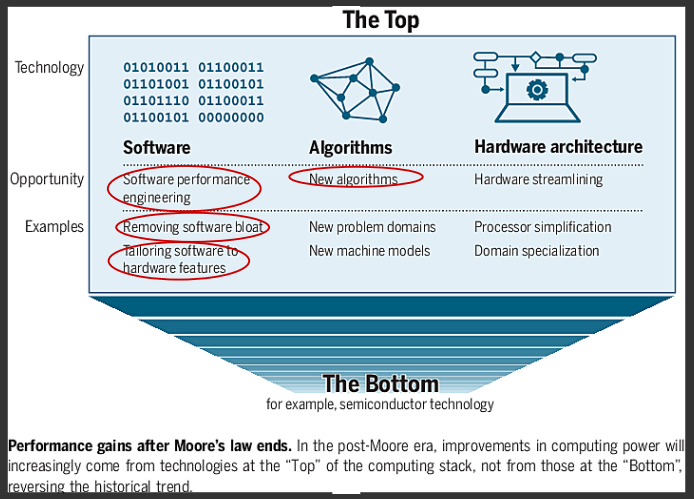


Filling the room at The Top

1. Removing software bloat







- Work in ARDG/ngCASA on C++ – Py interface (Jagannathan, Jan-W)

2. Tailoring software to h/w: HPG
3. Software performance engineering: Kokkos
4. Algorithms: Asp/Wasp, WiSClean: Reduce the number of the expensive Major Cycles
5. New Hardware: CEREBRAS? +New Algorithms
   Yet unknown tech/arch
6. Flexible s/w arch, with clearer separation of domains will be critical for success (adapt to h/w evolution without needing to re-write large parts of the code base).

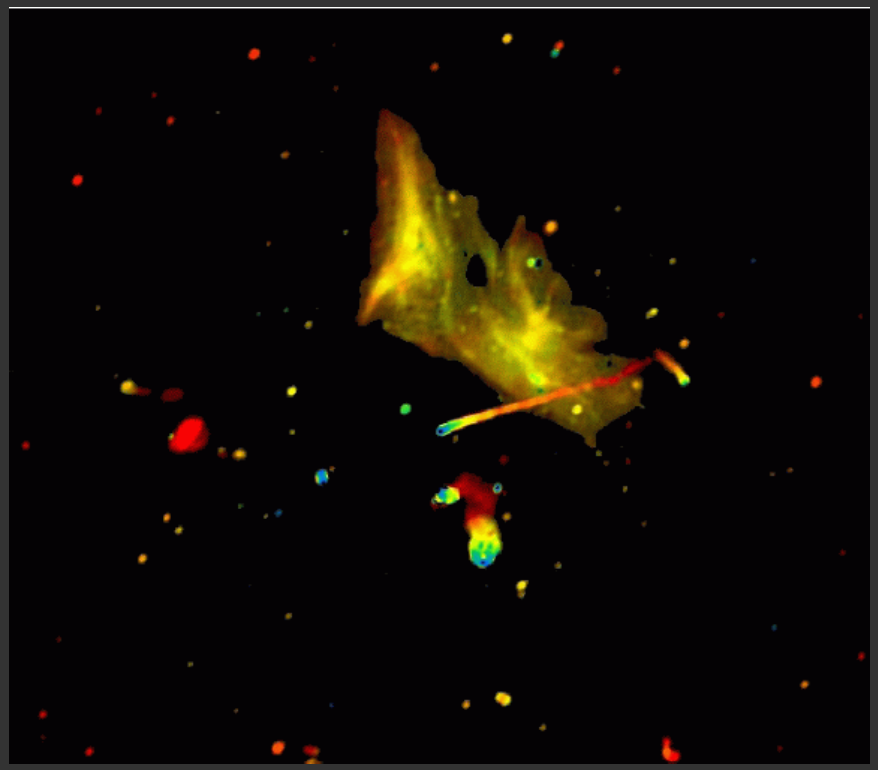
Algorithms/Software outlook

- Focus on Arithmetic Intensity of the algorithmic engines
  - Successful scaling will depend on efficient use of the silicon real estate
  - Algorithms tuned for telescope capabilities/peculiarities, Plug-in architecture
  - Hawk's Eye on complexity (computational, software, deployment,…)
- Heterogeneous computing will be essential
  - Needs flexible architecture which can be re-configured quickly and cheaply
- Performance engineering tools will be important
  - Reduces complexity: E.g. Kokkos: Front-end – Back-end design: Same code for CPU, GPUs,...
  - Prescriptive programming: Library- vs language-based approach to performance and portability
- Build scientific functionality from simpler components, minimize software bloat
  - Enables cross-discipline collaborations: Clear separation of RA/CS/HPC-domains
  - Various components deployed on geographically distributed, heterogeneous hardware
  - Simpler dependency graph
    - » Resolving dependency graph is a complex problem (formally unsolvable!). Simplicity helps (a lot!)

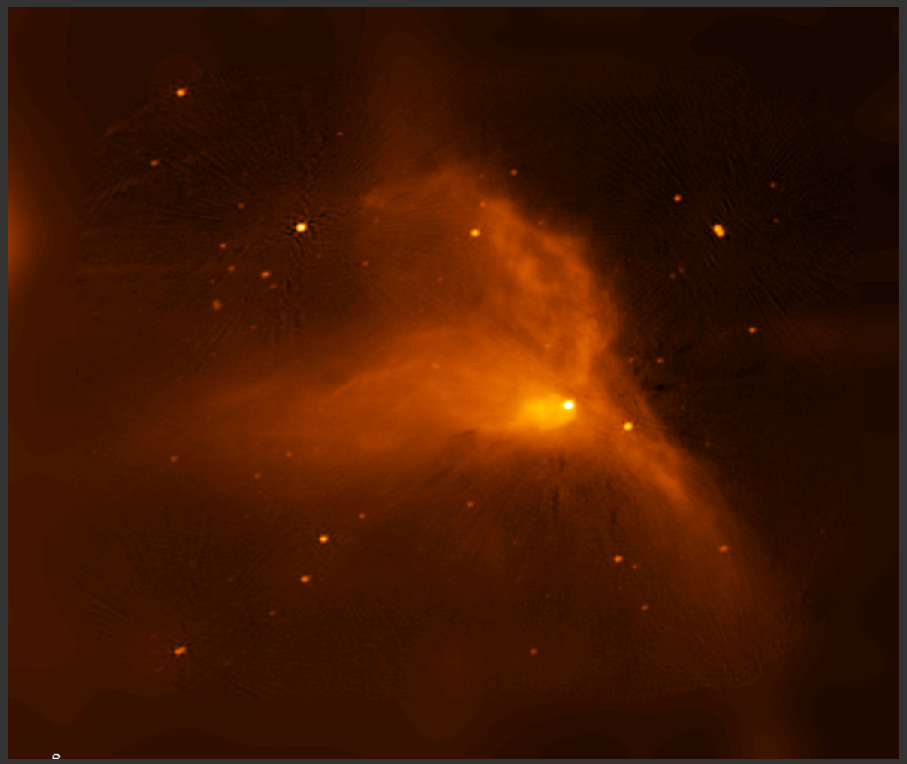
Algorithms/Software outlook

- Implications for design/planning:
  - Expect rapid technology evoluation: Scalable algorithms, scalable architecture needed
  - Operational efficiency: time(deployment) << time(Tech. evoluation)
  - Moore's era characterized by "Minimize software development time"
  - Now: Minimize run-time. Ease of programming a secondary driver
  - Core implementation in run-time performant languages
- "Software components: Only the giants survive."
  - Lampson; Theory, Tech., & Applications; Herbert, Jones Eds., (2004) pp 137
  - Implementation for specific problems, optimized for specific h/w
    Plugged-in in a higher-level (generic?) framework
  - Opposite of the "...no bespoke components...everything general-purpose..." mantra
- Paradigms appropriate for the past decade(s) may not be for the coming one(s)

Imaging with the EVLA @ L-Band



Single pointing, wide-band image (Rau, Owen)



Wide-band ~200 pointing mosaic+Single Dish