Implementation of NRAO's imaging workflow in HTCondor

Felipe R. H. Madsen - NRAO

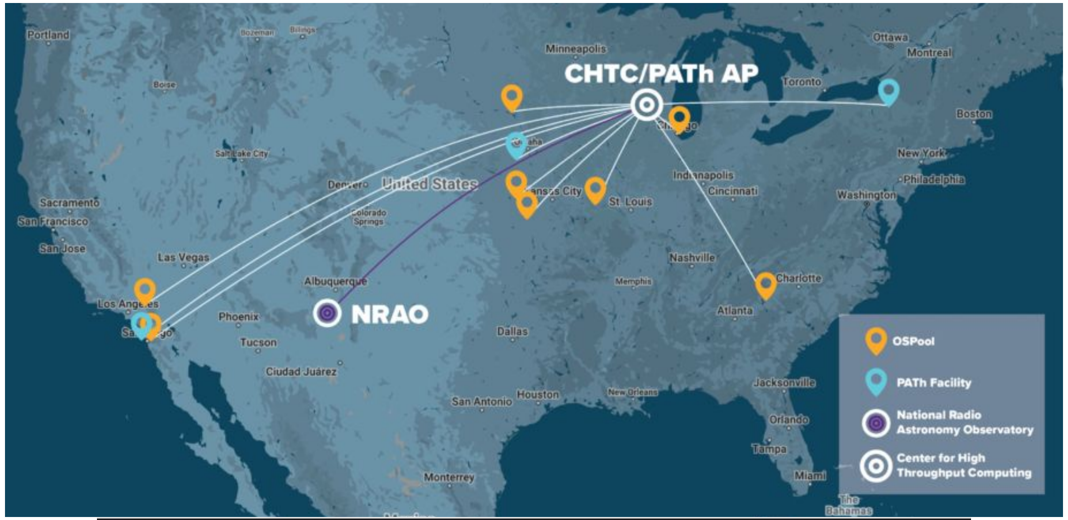
National Radio Astronomy Observatory

Abstract

A distributed workflow to make interferometric images was developed at NRAO and successfully deployed to nearly one hundred GPUs gathering resources from PATh and OSPool, to enable processing VLA data to make the deepest radio image of the Hubble Ultra Deep Field. We present the details of the HTCondor implementation of this workflow, as well as detailed results of performance metrics and how these results helped in identifying and prioritizing work on improvement opportunities.

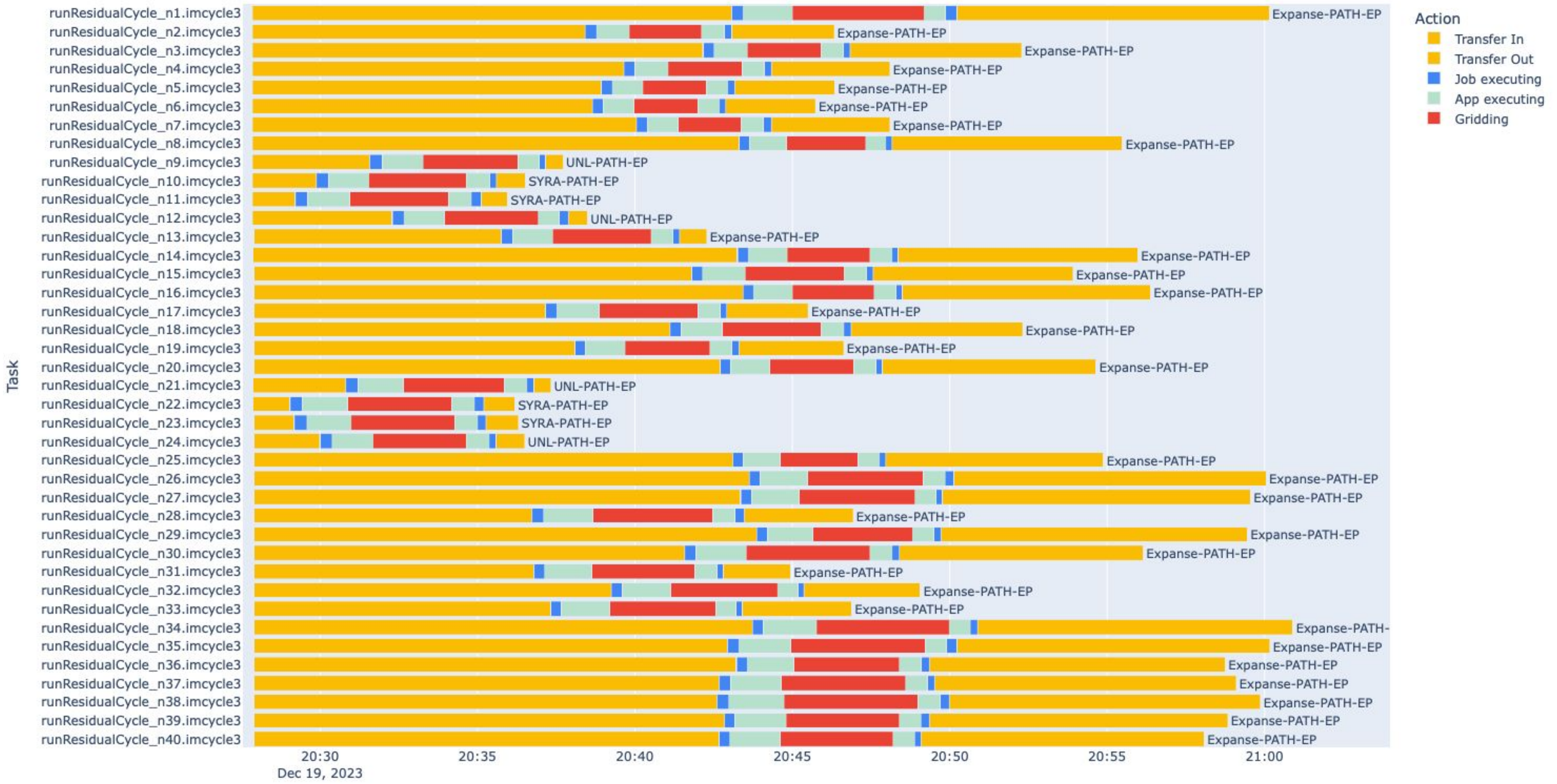
Some facts about December's imaging run

- Deepest radio image of the Hubble Ultra Deep Field
- 76 concurrent GPU jobs (peak)
- 2.7 jobs/minute, 1.5 TB/h (average of fastest cycle)
- ~ 750 total GPU hours
- 1650 jobs
- 5 days wall clock time, ~ 24 hours aggregate processing time



NSF AUI National Radio Astronomy Observatory

Concurrency plot



Analysis of the concurrency plots helps identifying and prioritizing improvement opportunities

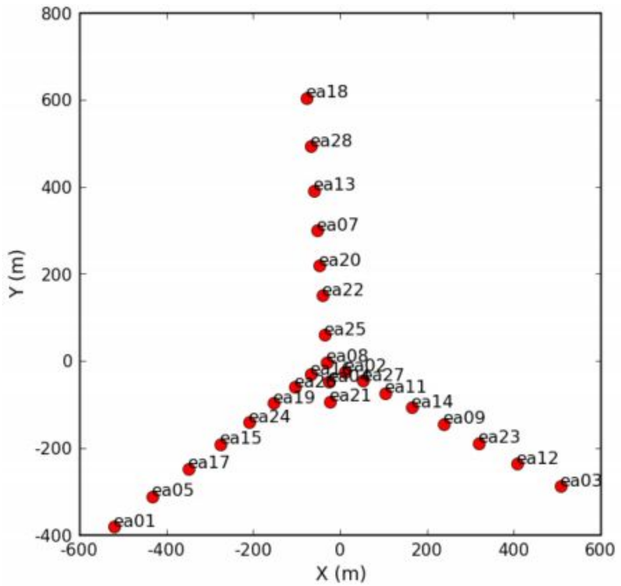
Future work / challenges from my talk on HTC23 (prioritized tasks completed/ongoing marked red)

- Improve IO efficiency of gridding jobs
- Further expand data partitioning to other axes
- Integrate/test new model cycle software module
- "Gather barrier"
  - Optimize DAG design to minimize barrier
  - Investigate scalable design solutions
- Scale up residual cycle partitioning / distribution by at least 1-2 orders of magnitude - will need larger number of available GPUs, or hybrid distribution model (GPU/CPU)

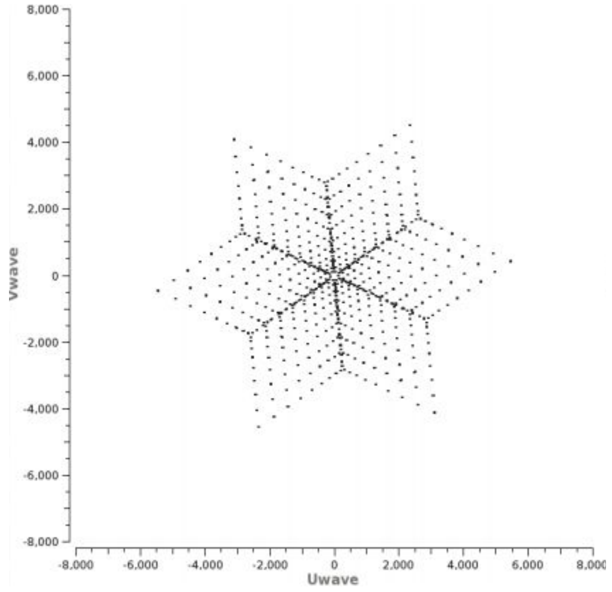
From the infrastructure's perspective, the concurrency plot can be a powerful tool to help identify sites or EPs with long transfer times, especially if combined with unit tests designed to monitor system performance.

Interferometric Imaging is a Computational Problem

Antenna positions (VLA)

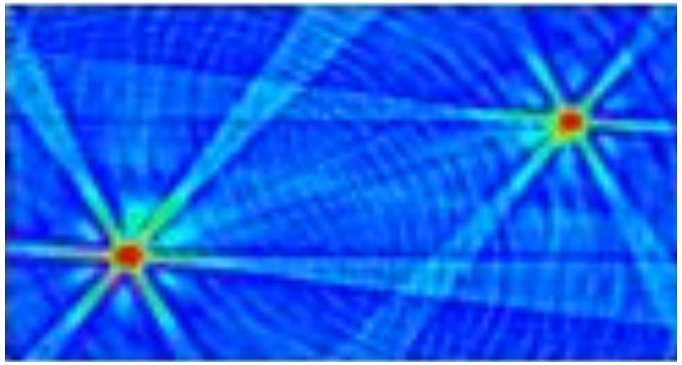


Distance, Orientation of all antenna pairs



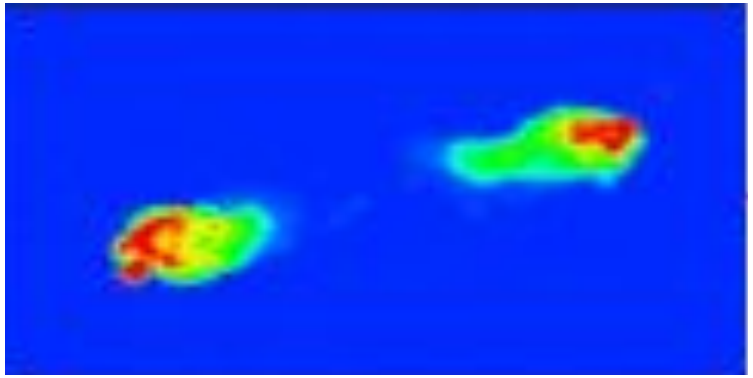
Sampling function

Gridding + FFT^{-1} of measurements



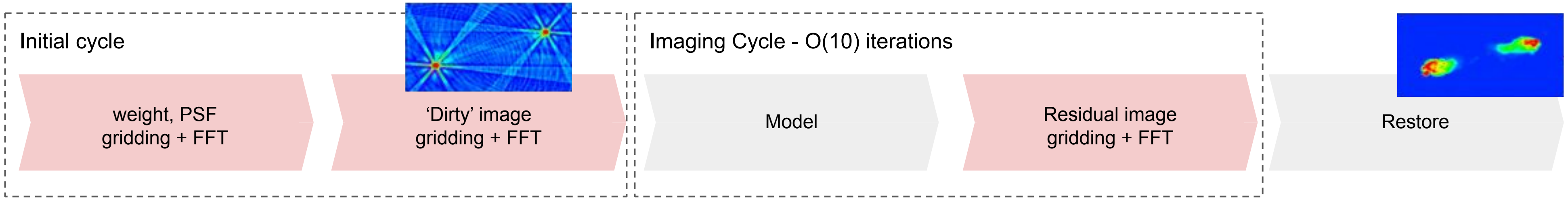
Dirty Image

Post processing



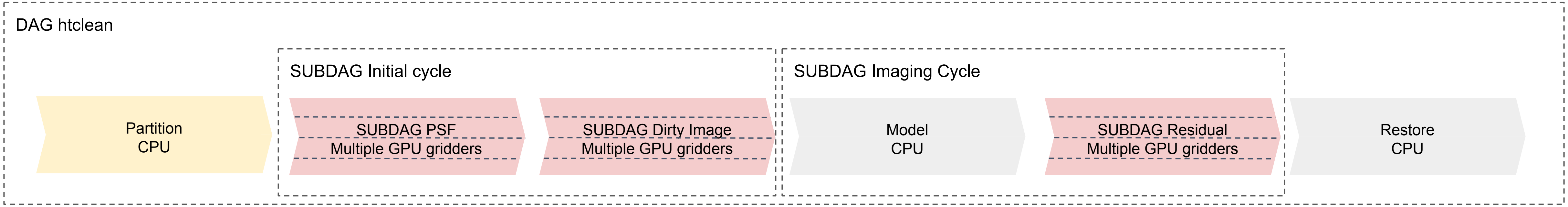
Final Image

Interferometric imaging workflow (CLEAN method)



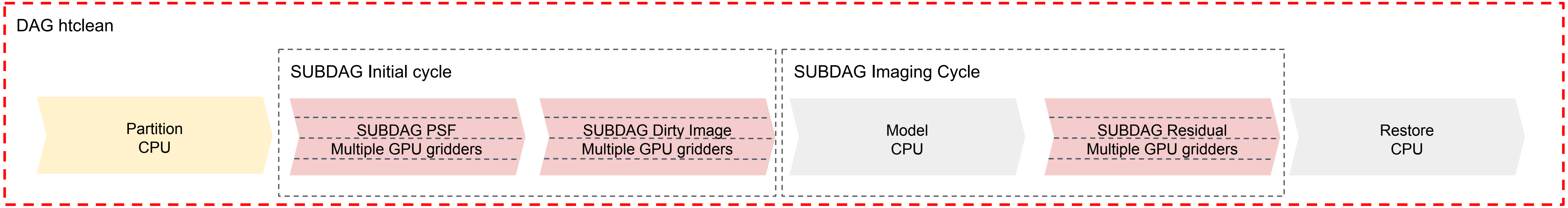
- Initial cycle
  - compute individual data weights and point spread function (PSF) for image reconstruction
  - compute 'dirty' image
- Imaging cycle - typically O(10) iterations
  - update Model: iteratively 'deconvolve' sampling function (PSF) and identify model components
  - update Residual: subtract model components from data and make new image
- Restore: combine PSF, model and residual to generate final image

HTClean implementation (HTCondor + CLEAN = HTClean)



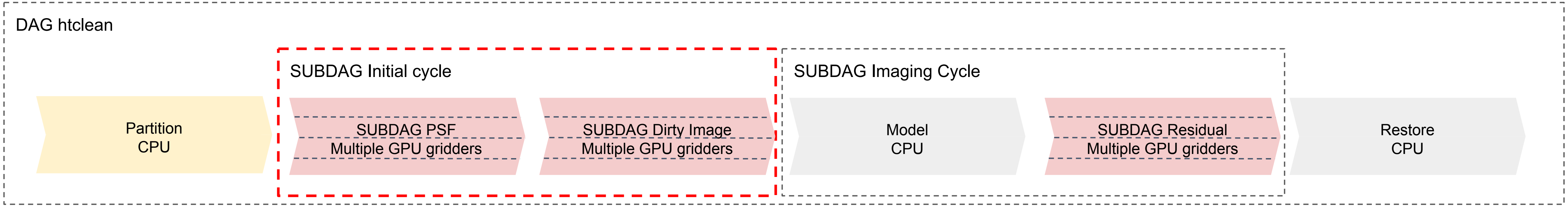
- early dev. 2020 (multiple CPU jobs) → 1st GPU job 2021 → multiple GPUs 2022 → deployment/development on PATh: late 2022 → Science run Dec 2023
- Nested DAGs with a RETRY+POSTSCRIPT to control iterations
- HTClean breaks down the imaging process in independent sessions that can be distributed
  - Addresses asymmetry in the two main stages of imaging:
    - residual cycle: highly parallelizable, high FLOPS, visibility domain
    - model cycle: serial (*continuum* imaging), low FLOPS, image domain

HTClean implementation - main DAG



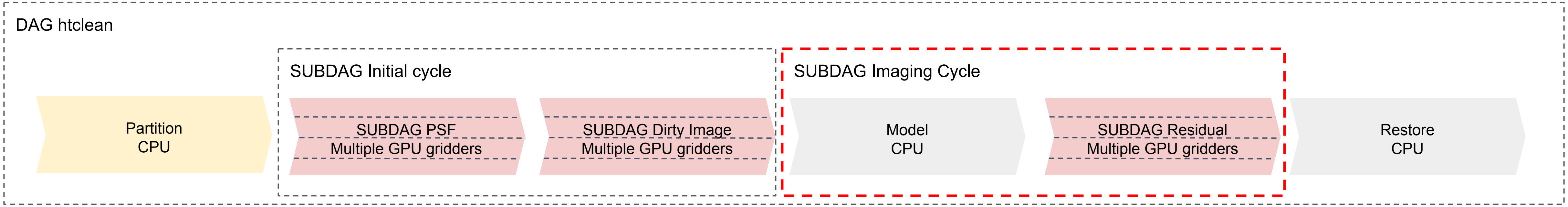

```
JOB              MSpartition      MSpartition.htc
SUBDAG EXTERNAL  initialCycle     initialCycle.dag
SUBDAG EXTERNAL  imagingCycle     imagingCycle.dag
JOB              makeFinalImages  makeFinalImages.htc

SCRIPT PRE       initialCycle     set_retry.sh -1
SCRIPT PRE       imagingCycle     set_retry.sh $RETRY
RETRY            imagingCycle     8
SCRIPT POST      imagingCycle     stopIterations.sh

PARENT           MSpartition      CHILD          initialCycle
PARENT           initialCycle     CHILD          imagingCycle
PARENT           imagingCycle     CHILD          makeFinalImages

VARS             ALL_NODES        jobmode="$(JOB)"
VARS             ALL_NODES        input_file="../bin/libra_htclean.def"
VARS             ALL_NODES        partId="n"
```

HTClean implementation - SUBDAG initial cycle



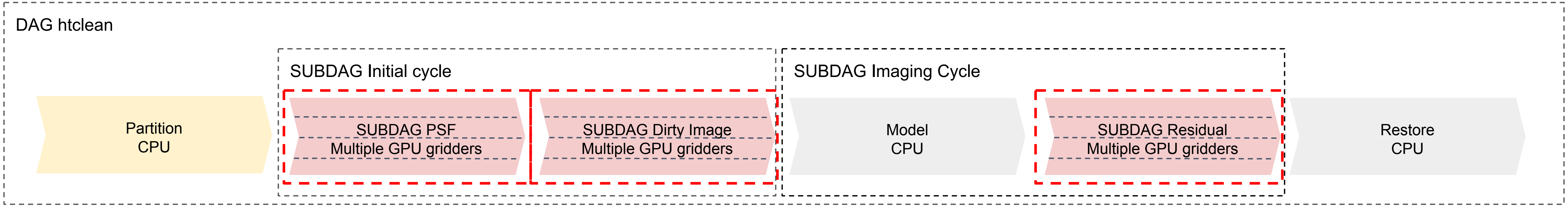

```
SUBDAG EXTERNAL   makePSF          makePSF.dag
JOB               gatherPSF        gatherPSF.htc
SUBDAG EXTERNAL   makeDirtyImage   makeDirtyImage.dag
JOB               gather           gather.htc

SCRIPT PRE        makePSF          writeGriddingDAGs.sh libra_htclean.def
SCRIPT PRE        gatherPSF        makeInputFileList.sh libra_htclean.def imcycle.htc psf
SCRIPT PRE        gather           makeInputFileList.sh libra_htclean.def imcycle.htc dirty

VARS              ALL_NODES        jobmode="$(JOB)"
VARS              ALL_NODES        input_file="../bin/libra_htclean.def"
VARS              ALL_NODES        partId="n"

PARENT            makePSF          CHILD gatherPSF
PARENT            makePSF          CHILD makeDirtyImage
PARENT            makeDirtyImage   CHILD gather
PARENT            gatherPSF        CHILD gather
```

HTClean implementation - SUBDAG imaging cycle




```
JOB              runModelCycle     runModelCycle.htc
SUBDAG EXTERNAL  runResidualCycle  runResidualCycle.dag
JOB              gather            gather.htc

SCRIPT PRE       gather            makeInputFileList.sh libra_htclean.def imcycle.htc residual

VARS             ALL_NODES         jobmode="$(JOB)"
VARS             ALL_NODES         input_file="../bin/libra_htclean.def"
VARS             ALL_NODES         partId="n"

RETRY            runModelCycle     2

PARENT           runModelCycle     CHILD runResidualCycle
PARENT           runResidualCycle  CHILD gather
```

HTClean implementation - Inner DAGs



- Inner DAGs are written dynamically according to input parameters

from initialCycle.dag:

```
SCRIPT PRE        makePSF            writeGriddingDAGs.sh libra_htclean.def
```

```
for i in `seq 1 ${nparts}`
do
    JOBtext+="JOB         ${DAG}_n${i}          ${DAG}.htc"$'\n'
    VARStext+="VARS        ${DAG}_n${i}          partId=\"n${i}\""$'\n'
    cfsize=`du -sh ${cfcachedir}/${cfcache}.tar | awk '{gsub(/G/, "", $1)}{print int($1)+1}'`
    mssize=`du -sh ${msdir}/${msname}_n${i}.ms.tar | awk '{gsub(/G/, "", $1)}{print int($1)+1}'`
    diskrq=$((2 * (cfsize + mssize) + 10))
    VARStext+="VARS        ${DAG}_n${i}          diskResidual=\"${diskrq} G\""$'\n'
done
VARStext+="VARS        ALL_NODES         jobmode=\"${jobmode}\""$'\n'
VARStext+="VARS        ALL_NODES         input_file=\"../bin/${input_file}\""$'\n'
VARStext+="RETRY       ALL_NODES         2"
TEXT=${JOBtext}$'\n\n'${VARStext}
echo "${TEXT}" > ${DAG}.dag
```

Dynamic disk allocation

Other highlights of the current implementation

Data transfers:

- auto-expand input data tarballs

```
transfer_input_files = $(cfcachedir)/$(cfcache).tar?pack=auto,
$(msdir)/$(msname)_$(partId).ms.tar?pack=auto,
$(imagesdir)/$(imagename).divmodel.imcycle$(imcycle).tar?pack=auto
```

Job configuration:

- detailed GPU requirements

```
+DESIRED_GPU_MIN_Capability = 8.0
+DESIRED_GPU_MIN_GlobalMemoryMb = 24000
+DESIRED_GPU_MIN_DriverVersion = 12.0
RequireGPUs = (Capability >= DESIRED_GPU_MIN_Capability) && (GlobalMemoryMb >
DESIRED_GPU_MIN_GlobalMemoryMb) && (DriverVersion >= DESIRED_GPU_MIN_DriverVersion)
```

- all jobs request singularity image

```
+SingularityImage = "/cvmfs/singularity.opensciencegrid.org/opensciencegrid/osgvo-el8:latest"
```

- unified parameter file for environment and imaging configuration
- all jobs use lightweight and specialized LibRA applications

Other highlights of the current implementation

GPU software execution:

- monitor GPU utilization

```
nvidia-smi --query-gpu=timestamp,name,utilization.memory,memory.used --format=csv -l 5 --id=${NVIDIA_VISIBLE_DEVICES} >
working/logs/nvidia.${jobmode}.${partId}.out &
```

- copy systems's libcuda.so.1 into application bundle

```
bundles_dir=`ls libra/bundles`
echo "bundles_dir: $bundles_dir"
if [ -e /.singularity.d/libs/libcuda.so.1 ]
then
    cp -f /.singularity.d/libs/libcuda.so.1 libra/bundles/${bundles_dir}/lib64
else
    if [ -e /lib64/libcuda.so.1 ]
    then
        cp -f /lib64/libcuda.so.1 libra/bundles/${bundles_dir}/lib64
    else
        echo "libcuda.so.1: File not found on knwon paths. Trying with packaged version"
    fi
fi
```

Future work / challenges

- Improve automation / reduce need of human intervention in running workflows:
  - Error: “Job credentials are not available” - fix deployed?
  - Applications and convolution functions currently pre-staged manually - is it a good use case for a Pelican origin?
  - Improve flexibility and scalability of data partition jobs
  - Deploy / integrate LibRA container