Kernel Prototyping SOW

Document Number……..……..... SKA-TEL-SDP-0000083
Type…….... REP
Revision……..... C
Author…… P. Braam, P. Wortmann
Release Date……..... 2016-07-04
Document Classification……..…….....……..... Unrestricted
Status……..……...…….. Released

Lead Author	Designation	Affiliation
Peter Braam		University of Cambridge
Signature & Date:		

Released by	Designation	Affiliation
Paul Alexander	SDP Project Lead	University of Cambridge
Signature & Date:		

Version	Date of Issue	Prepared by	Comments
.3	2016-07-08	Peter Braam	

ORGANISATION DETAILS

Name	Science Data Processor Consortium

Table of Contents

List of Figures

List of Tables

References

Applicable Documents

The following documents are applicable to the extent stated herein. In the event of conflict between the contents of the applicable documents and this document, **the applicable documents** shall take precedence.

Reference Number	Reference
AD-01	SKA-TEL-SDP-0000013 SDP Element Architecture Design
AD-02	SKA-TEL-SDP-0000015 SDP Execution Framework Design

Reference Documents

The following documents are referenced in this document. In the event of conflict between the contents of the referenced documents and this document, **this document** shall take precedence.

Reference Number	Reference
Legion	http://legion.stanford.edu/
Regent	http://regent-lang.org
Slurm	http://slurm.schedmd.com/

Audience:

1. SDP lead architects - for feedback on completeness and correctness
2. SDP - those supervising efforts (Jeremy Coles, Peter Braam, Peter Wortmann , Bojan Nikolic)
3. Industrial partners

Introduction

This document describes possible desirable directions SDP would like to see undertaken to study the performance of ***kernels*** on specific architectures.

The kernel we have selected is ***gridding***, because:

1. The memory bandwidth and processing gridding requires is very demanding
2. The data processed has domain specific characteristics
3. It is affected by numerous parameters

We separate the work that should be undertaken into the following sections:

1. Development of algorithms
2. Study of optimal data layout for cache re-use and processing
3. Alternative implementation of algorithms and data layout (e.g. sparse FFT)
4. Organization of the parameter space
5. Demonstrating the use of tools for profiling
6. Reporting the behavior for implemented algorithms

Description of the goals

Theory

At core, radio interferometry imaging is a Fourier transform operation, with most relevant effects mapping to simple multipliers at various stages. Data is gathered from the interference patterns (visibilities) of antenna pairs (baselines) which yields information about the sky brightness:

$$I(l,m) = \sum_{a_1,a_2 \in A} \int A_{a_1,t,f}(l,m) A_{a_2,t,f}(l,m) G_w(l,m) \left(\iint V_{w,a_1,a_2,t,f}(u,v) e^{2\pi i(ul+vm)} du\ dv \right) dw\ df\ dt$$

Here V are the input visibilities, which is a sum of delta functions, non-zero for every time and frequency when the u,v,w combination represents a baseline. In fact, the baseline coordinates are known in advance, as they are the coordinates of a line between the antennas, measured in wavelengths. Due to this definition and the earth's rotation this makes u,v,w a function of the involved antennas, time and frequency.

The Fourier transform of the visibility function of a single baseline yields the sky brightness pattern that corresponds to that single visibility. To bring all visibility contributions into a common image plane we further multiply by G_w to correct for baseline non-coplanarity effects and by antenna weights $A_{a,t,f}$ to correct for non-uniform antenna reception patterns.

To implement this efficiently, we would like to use fast Fourier transforms. However, we would not want to do repeat the FFTs twice as the formula above suggests. Fortunately, multiplication in image space is equivalent to convolution in frequency space, so we re-express the above equation as:

$$I(l,m) = \iint \left(\sum_{a_1,a_2 \in A} \int \left(\widehat{A}_{a_1,t,f} * \widehat{A}_{a_2,t,f} * \widehat{G}_w * V_{w,a_1,a_2,t,f} \right) (u,v)\, df\, dt\, dw \right) e^{2\pi i(ul+vm)} du\, dv$$

Where $\widehat{A}_{a,t,f}$ and $\widehat{G}_w$ are the Fourier transforms of the original functions. It can be shown that both functions quickly tend to zero for nonzero (u,v), which means that every visibility only contributes to a very small portion of the frequency plane. After summing up all these contributions, we only need to perform a single FFT to collect all visibility information into a single picture.

Example Code

We can easily implement imaging in just a few lines of Python using numpy. Note that for clarity this version is slightly simplified. Check the reference code for the full details including sub-grid coordinates.

GCF convolution

```
def aw_kernel_fn(theta, w, a1, a2, t, f):
      a1kern = a_kernel_fn(theta, a1, t, f)
      a2kern = a_kernel_fn(theta, a2, t, f)
      akern = scipy.signal.convolve2d(a1kern, a2kern, mode='same')
      wkern = w_kernel_fn(theta, w)
      return scipy.signal.convolve2d(akern, wkern, mode='same')
```

This computes the $\widehat{A}_{a_1,t,f} * \widehat{A}_{a_2,t,f} * \widehat{G}_w$ convolution of two A-kernels and one w-kernel. The parameter `theta` determines the grid resolution and depends on the imaged field of view. We leave the two kernel functions undefined here, their values should be considered input into the gridding algorithm.

Gridding

```
def convgrid(gcf, guv, p, v):
```

```
    gh, gw = gcf.shape
    x,y = numpy.floor(p + 0.5)
    guv[y-gh//2 : y+(gh+1)//2,
        x-gw//2 : x+(gw+1)//2] += gcf * v
```

The visibility value v gets multiplied by the grid convolution function and added at the appropriate position in the grid.

Imaging

```
def imaging(theta, lam, uvw, src, vis):
    N = int(round(theta * lam))
    guv = numpy.zeros([N, N], dtype=complex)
    for p, s, v in zip(uvw, src, vis):
        gcf = numpy.conj(aw_kernel_fn(theta, p[2], *s))
        convgrid(gcf, guv, theta * p, v)
    img = numpy.fft.ifft2(numpy.fft.ifftshift(guv))
    return numpy.real(numpy.fft.fftshift(img))
```

The parameter `lambda` is the size of the uv-grid, corresponding to the output image resolution and the length of the longest baseline. Together with `theta` it is used to determine the size of the grid to allocate. The parameters `uvw` and `src` list of the values of u,v,w and a_1,a_2,t,f where V is nonzero, with `vis` giving the visibility function values at these positions.

Once we have the grid allocated, we can use the kernel function to determine the convolution for every visibility and grid it at the appropriate position. After the grid is completely filled, we can use an inverse FFT to obtain an image. Note that our grid and image have the zero frequency/position in the center, so we need to shift before and after the FFT.

Optimisation Opportunities

The gridding operation is mathematically simple. However, the data and its representation is involved and will affect the performance.

Data properties influence the computation in many ways. The concurrency in computing the uv-grid should be apparent in the code shown above. The pattern baselines in the UVW grid is sparse, irregular, but computable. When V~ is computed for a particular time and frequency the result V~ may be nonzero on a sparse subset of the UV grid. When averaging V~ over frequencies (so called continuum imaging) and time, the support of V~ may not be sparse.

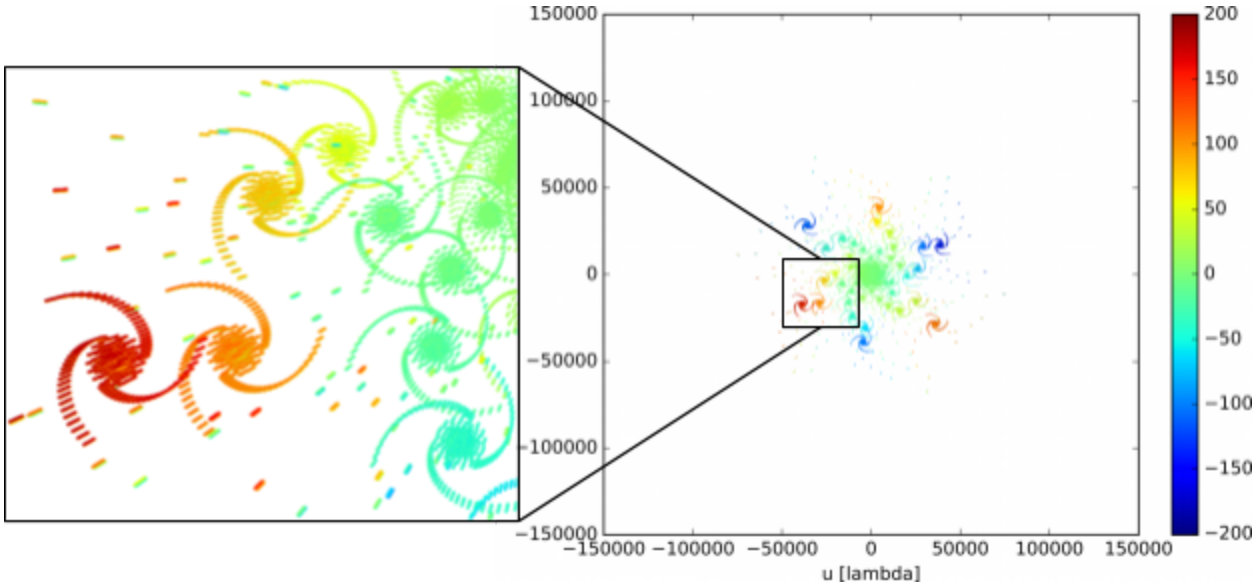

Figure 1: uv-grid coverage

The following considerations have not been studied in sufficient detail, and leave room to modify algorithms currently in use:

1. **Data Layout**. The visibility data at any given time are nonzero on a sparse subset of the u,v plane, with greatly varying density of samples when mapped to the u,v,w grid. Figure 1 shows the distribution that can be expected: A lot of empty space, a densely filled (but computationally comparatively cheap) center region and multiple “arms” for longer baselines. Note that longer baselines have larger w-values due to the earth’s curvature.

   Visibilities belonging to the same baseline snapshot cluster together closely, and can be seen on the left of Figure 1 as small lines. Creating appropriate “buckets” of visibility data exhibiting locality might help with caching. The resulting function on the u,v grid may be sufficiently sparse to warrant a sparse encoding.
2. **Concurrency**. Visibilities from different baselines may contribute to non-zero values of the gridded visibility at the same u,v grid points, and when they do so, the value at this gridpoint becomes data on which concurrency must be considered. Handling this appears to be computationally costly. However, it is likely that such concurrency may be avoided by clustering the computations - either by exploiting locality properties within the data (e.g. baseline, time and frequency) using the natural baseline structure of the data or using a checkerboard pattern of buckets.
3. **Sparseness**. The results from the gridding operation at a particular time are nonzero only on a sparse subset of the u,v grid. Appropriate data representation might lead to a smaller memory footprint.
4. **GCF locality**. Many different GCF’s will be used as the w-value, frequency channels, time and baseline vary. However, the usage scope of a kernel in the uv-grid is generally fairly local, which suggests that convolving them on-the-fly might be good idea. A model that relates the observed required memory capacity, cache misses associated with

changing GCF, and recomputation costs to the values found in the parametric model is valuable.

5. **Conjugated Visibilities.** Every visibility has an associated conjugated visibility with negative u,v,w coordinates. For this reason visibility data only contains one of them, and the hermitian property is restored before or in the FFT step. This means that the implementation can choose whether to grid the original or the conjugated visibility, which can be used to increase or decrease locality as appropriate.

Test Data

The visibility data reflects what a typical SKA1 Low snapshot will look like from the point of view of a gridding kernel. Visibility data and kernels will have the following characteristics:

- 512 stations, therefore 130816 baselines
- Snapshot length 45 s, with 50-5 time steps (split depends on baseline)
- 7.5 MHz frequency range, with 300-20 channels (again, depends on baseline)
- just 1 polarisation

Furthermore, we assume the following gridding configuration:

- All kernels have size 15x15
- w-kernels are oversampled by a factor of 8
- A-kernel scopes are all 10s and 0.9MHz
- Theta (field of view dimension): 0.08 radians
- Lambda (uv-grid dimension): 300000 wavelengths
- Grid resolution: 0.08 * 300000 = 24000 uv-cells on each side

Visibilies

http://www.mrao.cam.ac.uk/~pw410/crocodile/SKA1-LOW_v6_snapshot_vis_hi/vis.h5.gz
[1.4GB, 43930418 visibilities]
http://www.mrao.cam.ac.uk/~pw410/crocodile/SKA1-LOW_v6_snapshot_vis_hi/quick.h5.gz
[9MB, 130816 visibilities]

Visibility data is packaged as an HDF5 file. The full data set contains a group "`vis/[a1]/[a2]`" for every antenna pair (and therefore baseline), whereas the quick dataset only contains one group of visibilities. Each visibility group is laid out as follows:

- `frequency {nfreq}`: List of frequencies (double, Hz)
- `time {ntime}`: List of timeslots (double, UTC MJD)
- `uvw {ntime x 3}`: List of baseline coordinates (double, in metres)
- `vis {ntime x nfreq x npol}` : Visibility data (complex double)

Baseline coordinates in wavelengths can be calculated by multiplying by frequency and dividing by the speed of light.

W-kernels

W-kernels [60MB, 268 w-planes, 1.5 lambda spacing]

W-kernels are similarly grouped in an HDF5 file. There will be groups "`wkern/[theta]/[w]`" for kernels suitable for gridding a certain field of view at a given w-value. Each group will contain the data set:

- `kern {nover x nover x nsupport x nsupport}`: Kernel data (complex double)

The w-kernel to apply to a visibility is the one with the closest w-value present in the kernel set.

A-kernel format

A-kernels [247MB, 512 antennas, 10s, 0.9 MHz]

For A-kernels we will have groups "`akern/[theta]/[a]/[t]/[f]`" for A-kernels suitable for a certain antenna, time and frequency. Again there will be just the kernel data set:

- `kern {nsupport x nsupport}`: Kernel data (complex double)

When in doubt, the kernel with the closest time and frequency should be selected.

Reference Code

https://github.com/SKA-ScienceDataProcessor/crocodile

The reference implementation is going to be the ARL (aka crocodile[1]). The goal is to implement a more efficient version of gridding (`w_cache_imaging`). This includes all functions called from this function, including the callbacks to cache Aw-kernel convolution and cache functions. All of these should be considered relevant for performance and should be benchmarked together appropriately.

Input data can be pre-processed as much as necessary to achieve an efficient implementation, but the overhead of doing so will need to be characterized. Creating overlapping I/O and computation is expected to be beneficial at least in some ranges of the parameters values. Data can be reorganised into any layout. Furthermore, arbitrary preparation steps are allowed on all data **except** the actual visibility values.

The objective for the kernel is to produce a result that would be equivalent to the grid output of the reference implementation after applying `make_grid_hermitian`. This means that for every visibility, the implementation can choose freely to either grid it unchanged or conjugated (negate u, v and w coordinates and complex-conjugate the visibility).

[1] http://www.phy.cam.ac.uk/history/years/croc

Examples

Quickest - quick data set, simple imaging:

```
scripts/image_dataset.py --theta 0.08 --lambda 300000
                         --grid out.grid --image out.img
                         quick.h5
```

Full data set, using only w-kernels:

```
scripts/image_dataset.py --theta 0.08 --lambda 300000
                         --grid out.grid --image out.img
                         --wkern wkern.h5 vis.h5
```

Full data set with Aw-kernels:

```
scripts/image_dataset.py --theta 0.08 --lambda 300000
                         --grid out.grid --image out.img
                         --wkern wkern.h5 vis.h5
```

Expected Results

The image result will look very similar irrespective of gridding method, as visibilities have very low w-values, and for the purpose of this test the A-kernels only adds noise.

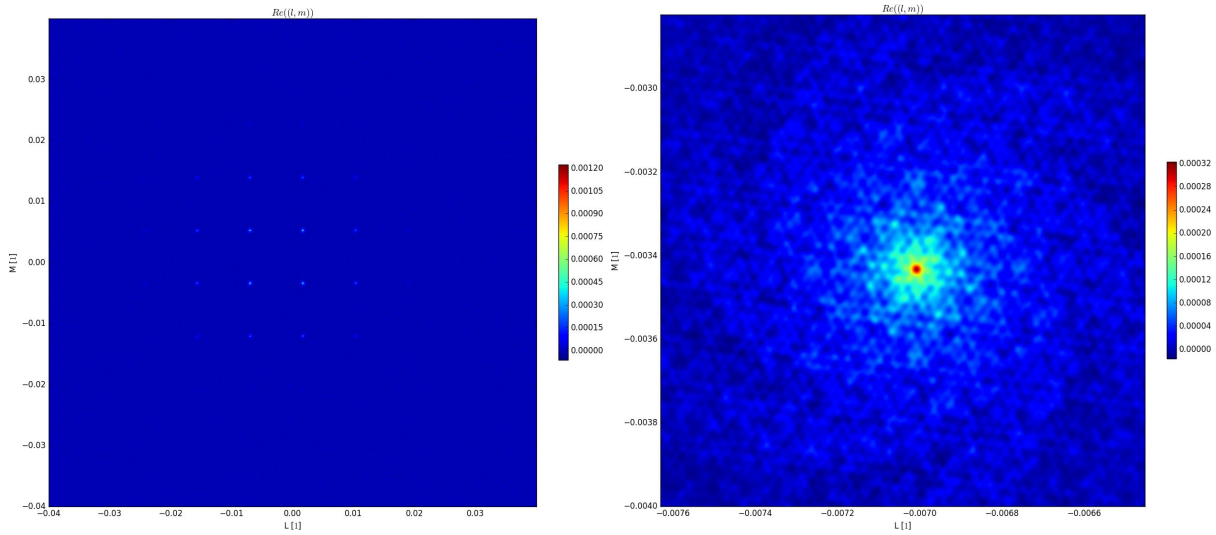

Figure 2: Image result for w-projection of the “quick” visibility set

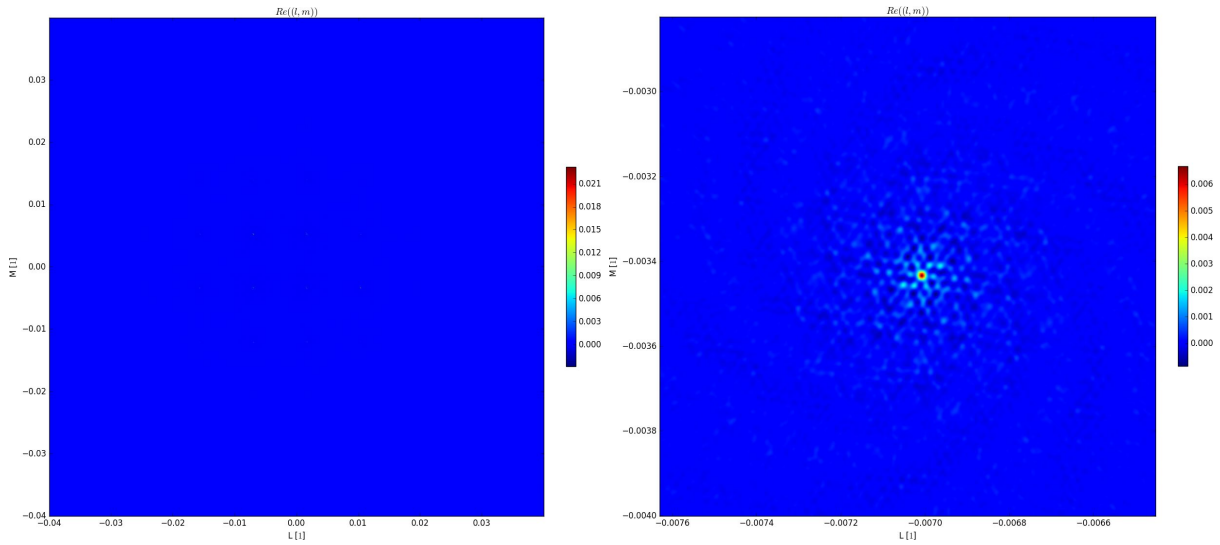

Figure 3: Image result for w-projection of the full visibility set

For both data sets a grid of points should be visible, with points 0.5 degrees (0.008726 radians, 2618 pixels) apart. Note that while the points are more apparent in the overview of the “quick” dataset, the signal is actually much sharper and stronger for the full data set, as the right-side pictures show.

Reference data results can be obtained here:

	Simple imaging	w-projection	Aw-projection
“Quick” dataset	Grid (md5), Image (md5)	Grid (md5), Image (md5)	Grid (md5), Image (md5)
Full dataset	Grid (md5), Image (md5)	Grid (md5), Image (md5)	Grid (md5), Image (md5)

All grid and image files are 24000 x 24000 arrays of double-precision complex numbers and double-precision real numbers respectively, in row-major order.

Profiling Toolchain

One of the most important tasks to be undertaken is the use and possible development of a tool chain to understand exactly why particular algorithms perform in a particular way. For every kernel developed we would like to understand:

1. What % of system maxima are achieved for FLOPS/sec, bytes/sec (memory bandwidth), bytes/sec (IO bandwidth), cache re-use, energy consumption? A direct comparison must be made with

    a. The parametric model, as it is updated to account for increased efficiency (e.g. using computed vs imported baselines, compression, precision, etc.) Note that some variations, e.g. the use of Fourier transforms for sparse data are not incorporated in the parametric model at present.
    b. The theoretical performance of the system
2. What is limiting the performance? Here we need detailed information, e.g. the dominating factor is data movement in ***stated instructions in the kernel*** from RAM to L3, it achieves xxx B/sec, % of max, or atomic operations on the following arguments dominate the computation time.
3. What overhead in the observed computation is attributable to synchronization to handle concurrency?
4. To what degree is the caching hierarchy exploited optimally?
5. Document the dependency of performance when varying numerical parameters over a wide range and explain such dependencies. Numerical values that can be varied include:
    a. Grid sizes (note: kernel sizes are affected by grid sizes), performing the computations with the baselines with upper and lower bounds on length.
    b. Convolution function support sizes
    c. Threads started, cores utilized
    d. Frequency and number of channels processed
6. Document and explain the dependency on non-numerical parameters, e.g.:
    a. Different sorting and grouping of visibility data
    b. Programming models employed (MPI, vs OpenMP, CUDA, others)
    c. Different data representations (e.g. float vs double, compressed, utilizing lookup tables)

Deliverables

The following deliverables exist in connection with a prototype kernel:

1. Files with performance data that was evaluated.
2. All input sets, runtime and system configurations including code. Results that cannot be reproduced are not valuable.
3. Transient observations and explanations, prepared for maximum re-usability, e.g.:
    a. It was observed that communicating MPI processes (using the ABC MPI implementation version X.Y.Z) delivered better performance than communicating POSIX threads (using libc version U.V.W), because MPI communication has an optimization ….. not found in the thread implementation.
    b. The memory bandwidth that was achieved was XX% of the maximum, due to ….
4. Graphical representations of key findings underlying point 3. If aspects of the parameter space are not explored, state this clearly and state why the results remain relevant.

We recommend that a shared ***SDP kernel performance study*** document must be maintained summarizing important conclusions with references to deliverables.