Discussion Document: SDP Compute Platform Requirements

Document number ..... Not Defined
Context ..... Not Defined
Revision Draft-a502d04
Author ..... B. Nikolic
Release ..... Not released
Date ..... Not released
Document classification ..... Unrestricted
Status ..... Draft

Name	Designation	Affilitation
B. Nikolic	Lead Author	Astrophysics Group, Cavendish Laboratory, University of Cambridge
Signature & Date:		
Name	Designation	Affilitation
Signature & Date:		

Version	Date of issue	Prepared by	Comments
0.1	2016-05-16	B. Nikolic	Initial circulated version

ORGANISATION DETAILS

Name	Science Data Processor Consortium

1 Introduction

The purpose of this document is to set the scene for technical discussion about the SDP Processor Platform. The approach is:

1. Identifying the general, high-level questions that the Platform design needs to address (Section 2)
2. Key requirements . These are developed from basic top-level requirements through to some lower level requirements by systematically introducing assumptions/ potential design options. The requirements are quantified where ever possible but the numerical values should all currently understood to be indicative.
3. Some further more detailed questions (Section 4). The purpose of these is to help with further analysis of requirements in future iterations of this document.

2 High-level questions

1. We think one of the main limitng factors for current SKA/SDP algorithms and CPU/GPU architectures is memory bandwidth. What is the most power/cost efficient way to maximise memory bandwidth of the SDP processor platform?
2. How do we meet the demanding visibility buffer read requirements? What is the cost/power/reliability impact?
3. Should we aim for data locallity for the visibility data? Possibilities include:
   (a) Intermingled storage and compute on node level (with or without data locallity)
   (b) Intermingled storage and compute at rack level (with or without data locallity)
   (c) Separate storage and compute (no possibility of data locality)
4. Are disk based visibility buffer storage solution viable? (High capacity of disks might be an advantage if they are indeed viable, see Section 3.25)
5. How do we ensure the read latency for data from the visibility buffer does not reduce efficiency of processing?
6. How do we load-balance the SDP processing?
   (a) Different observations can have different (by orders of magnitude) computational requirements for processing
   (b) Processing of each observation is divided into stages with potentially different parallelisation strategies, resource utilisation patterns
   (c) Tasks within each stage have significantly different resource needs
   (d) Lots of possibilities for stochastic effects
7. How do we get adequate recovery from failures?
8. Programming model/language/run-time at various granularity:

(a) Fine granularity shared memory (OpenMP/CUDA/OpenACC/pthreads/?)

(b) Within task distributed memory (MPI/PGAS/?)

(c) Task parallelism (data-driven/dataflow ?)

(d) Observation level (a big data scheduling system?)

9. (Communication between tasks: can we replace (large) message passing with storage-like semantics?)

3 Requirements

Notes on these requirements: They begin with top-level requirements; then further requirements are identified by adding some assumptions/potential design choices. Introducing some putative assumptions/design choices is necessary in order to develop the requirements to a lower level. Comments on all of these are welcome as well as alternative assumptions and design choices.

3.1 Top-level Requirements

These requirements are derived from:

1. The data rates input specified to be into the SDP from the SKA Correlator
2. Assumptions about the processing required to process these data into scientifically useful images
3. A pen-and-paper analysis counting the number of numerical operations required to perform the processing

No assumptions about the compute platform is used in their derivation and they are therefore independent of its architecture.

3.2 Floating Point Operations Throughput

This is the *average achieved* rate at which floating point operations are performed:

$$R_{\mathrm{FLOP}} > 100\,\mathrm{PetaFLOP\,s^{-1}} \tag{1}$$

An analysis of how much of this (if any) can be done single-precision has not yet been done so currently we assume *all operations are* **double** *precision*.

3.3 Visibility buffer throughput

This is the average achieved visibility buffer read and write rates:

$$R_{\mathrm{buf}}^{\mathrm{write}} > 1\,\mathrm{TeraByte\,s^{-1}} \tag{2}$$

$$R_{\mathrm{buf}}^{\mathrm{read}} > 10\,\mathrm{TeraByte\,s^{-1}} \tag{3}$$

The reads and writes are expected with large block sizes, e.g., one megabyte or more.

3.4 Minimum visibility buffer size

The visibility buffer needs to hold *at minimum* two observations at one time and also intermediate data produts. Taking into account visibility data volume, further metadata and a margin this gives approximately:

$$M_{\mathrm{buf}} > 50\,\mathrm{PetaByte} \tag{4}$$

3.5 Data input into the SDP

The data will arrive to SDP in UDP packets on multiple 40 GbE links without flow control. The data rate in is $0.5\,\mathrm{TeraByte}$. It is acceptable to drop a small fraction of the incoming packets.

3.6 Assumption: operational intensity

The key kernels of the SDP processing have been profiled on current high-end CPU and GPU architectures. It is found that with close to optimal algorithmic choices, both architectures the maximum operational intensity is around 0.5 FLOP/Byte. See the short survey by Nikolic (2016a) for references. This is an estimate derived using current known alogirhtms and for current processor architectures.

Based on this we assume operational intensity $\rho_{\mathrm{op}} = 0.5\,\mathrm{FLOP\,s^{-1}}$.

3.7 Aggregate memory bandwidth requirement

Using Section 3.6, assuming there aren't large changes in processor architectures we can also assume the maximum operational intensity will remain constant, giving an aggregate memory bandwidth for the system:

$$R_{\mathrm{mem}} > 200\,\mathrm{PetaByte\,s^{-1}} \tag{5}$$

This is the bandwidth to the high-bandwidth working memory.

3.8 Assumption: distributed memory operation

We assume the key kernels will work in distributed memory fashion with distribution over N_{cu} equal-sized compute units. The justification for this is that it would be very difficult to achieve throughput with shared memory operation on this scale.

This gives a relation ship for per-compute unit operation throughput and memory bandwidth:

$$R_{\mathrm{FLOP}} = N_{\mathrm{cu}} R^{\mathrm{cu}}_{\mathrm{FLOP}} > 100\,\mathrm{PetaFLOP\,s^{-1}} \tag{6}$$

$$R_{\mathrm{mem}} = N_{\mathrm{cu}} R^{\mathrm{cu}}_{\mathrm{mem}} > 200\,\mathrm{PetaByte\,s^{-1}} \tag{7}$$

3.9 Assumption: degree of facetting

The natural working memory size for an SDP computation is a grid around $10^5 \times 10^5$ in size, with each cell a complex double-precision number and two such grids are usually needed. However the technique of 'faceting' can reduce this working memory size up to a certain limit. A discussion of possible limits is presented by Nikolic (2016b).

Here we will assume maximum degree of facetting is 6×6.

3.10 Working memory size

Using assumptions Sections 3.8 and 3.9 the minimum working memory per compute unit can be estimated as:

$$M^{\mathrm{cu}}_{\mathrm{mem}} > 9\,\mathrm{GigaByte} \tag{8}$$

This defines the minimum size of the high-speed hing-bandwidth memory attached to each compute unit.

3.11 Assumption: high-bandwidth memory architecture

We next assume high-bandwidth memory architecture for the compute units. We assume 256 GB/s memory bandwidth per stack, 4 GigaByte capacity per stack, $E_{\mathrm{mem}} \sim 6\,\mathrm{pJ\,bit^{-1}}$ energy to access each bit.

3.12 Minimum size of compute units

The assumptions in Section 3.11 allow show that at least three HBM stacks that need to be attached to each compute unit. Since it is likely an even number will be used, we therefore assume 4 stacks per compute unit.

The HBM architecture summed in Section 3.11 then gives:

$$M^{\mathrm{cu}}_{\mathrm{mem}} \sim 16\,\mathrm{GigaByte} \tag{9}$$

$$R^{\mathrm{cu}}_{\mathrm{mem}} \sim 1\,\mathrm{TeraByte\,s^{-1}} \tag{10}$$

Relationship in Equation (7) implies the useful floating point capacity should be about

$$R^{\mathrm{cu}}_{\mathrm{FLOP}} > 500\,\mathrm{GigaFLOP\,s^{-1}} \tag{11}$$

3.13 Power estimates

The energy-per-bit access of Section 3.11 allows computational system power estimates.

It can be seen that energy use associated with memory access will likely dominate the total memory budget in a throughput-designed compute unit:

1. Lucas et al. (2014) estimates $12\,\mathrm{pJ\,bit^{-1}}$ energy is required for the actual FLOP calculation in 28 nm silicon
2. The recently announced[1] NVidia Telsa P100 is estimated at around $5000\,\mathrm{GigaFLOP\,s^{-1}}$ in about 250 W giving total of $50\,\mathrm{pJ\,bit^{-1}}$ combined for calculations and memory access as well as all other functions.

In contrast .5 FLOP/Byte operational intensity (Section 3.6) implies 16 bits of memory transfer per FLOP which for HBM can be estimated as around $100\,\mathrm{pJ}$ (see 3.11), showing that memory transfer will dominate the power budget of computational units.

3.14 Number of compute units to meet computational requirement

Given Equation (7) and Section 3.12 we can estimate the maximum number of computational units needed as $N_{\mathrm{cu}} \sim 2 \times 10^5$.

[1] See for example `https://devblogs.nvidia.com/parallelforall/inside-pascal/`

3.15 Number of compute units to meet power cap

The power estimation based on memory bandwidth (see above, and also Nikolic 2016a) suggest power cap will limit the number of nodes. If we assume $P_{\max} \sim 4\,\mathrm{MW}$ is devoted to computation then the number of compute units, and that maximum $\eta_{\mathrm{mem}} \sim 1/3$ of energy in compute units can be devoted to the memory access then number of compute units is estimated as:

$$N_{\mathrm{cu}} = \frac{P_{\max}\eta_{\mathrm{mem}}}{R^{\mathrm{cu}}_{\mathrm{mem}} E_{\mathrm{mem}} 8\mathrm{bits\,byte}^{-1}} \sim 3 \times 10^4 \tag{12}$$

3.16 Compute Node Architecture

If define a compute node as $K \geq 1$ compute units packaged together, with a single main memory address space and acting as a single network end-point.

$$N_{\mathrm{node}} = N_{\mathrm{cu}}/K \tag{13}$$

The memory address space on a compute *n*ode is not assumed fully cache coherent unlike for a compute unit which is assumed fully cache coherent.

3.17 Network end-points

The maximum number of network end points for a non-power capped system is $N_{\mathrm{node}} \sim 2 \times 10^5$ while for example for $K = 4$ with compute units twice the minimum size and a a system sized to meet the power cap the $N_{\mathrm{node}} \sim 4 \times 10^3$.

3.18 Assumptions: network architecture possibilities

There are two obvious possible connectivity architectures:

1. For end-points at the low end given in Section 3.17 a fully non-blocking network may be possible
2. Otherwise some sort of tree topology is likely to fit the problem best as the main dimension of parallelisation is the one-dimensional space of frequencies.

3.19 Injection bandwidth past first-level switches requirement for intermediate data products

The network bandwidth required to communicate the intermediate data products is difficult to extimate. The most comperhensive analysis is Peter Wortmann[2]. This analysis shows that without careful planning the data rates could be high but that with careful arrangements the achieved average injection bandwidth may need to be only 1-2 TeraByte/s. If racks are defined as the first level switches then the injection bandwidth into the upper layers of the network must correspondigly be at least 2 TeraByte/s.

[2] Slides and source code : see `http://www.mrao.cam.ac.uk/~pw410/sdp-par-model/data-flow-presentation.pdf` and `https://github.com/SKA-ScienceDataProcessor/sdp-par-model/blob/master/iPython/SKA1_Dataflow.ipynb`

3.20 Network bisection bandwidth

Difficult to estimate. However if the visibility data is not localised to where it will be processed then approximately the bisection will need to be at least the read bandwidhth of the visibility buffer (10 TeraByte/s).

3.21 Failure recovery by major cycle check-point

The natural check-point for the main processing is the end of a major cycle. Check-pointing here is likely to be free or nearly so because saving state at end of major cycle is natural part of the processing and the state size is unlikely to be very large. Major cycles are expected to be completed around every 30 minuts. So, if mean time between a failure requiring check point restart is 12 hours then average fraction of compute time lost is 15mins/12 hours, i.e., around 2%.

3.22 Parallelism overview

A hirearchy of parallelisation oportunites can be identified:

1. Receiving/processing multiple observations ("capabilities" in SDP language). This is embarasingly-parallel task-based parallelism with no interaction between the tasks at all (but shared computing resources may need to be used).
2. Intra-observation data- and task-based parallelism. The frequency of interactions between the tasks is low although the data volume exchanged between tasks may be significant.
3. Intra-task distributed memory strongly coupled parallelisation. This may be necessary for the calibration solver or image-plane clean for example. Here message frequency is likely higher, data volumes lower and there is stronger coupling between the computations.
4. Itra-task shared-memory parallelism, most likely SIMD+Threads or SIMT

3.23 Minimum Degree of shared memory paralellism

If we assume that one FPU has a throughput of one GigaOperation/second and taking into account the mininum compute unit size of Section 3.12 we can see that a minimum of 500-way shared memory parallelism will need to be supported by the hardware and software. This could be SIMD+Threads or SIMT or other shared memory execution models.

3.24 Load balancing intra-observation

The data-parallel processing task *do not* all naturally same execution time, e.g., the low-freuencies tend to require more processing than high frequencies. Therefore load-balancing is non trivial even in absence of the inevitable stochastic variation in O/S, node, storage and network performance which will introduce stochastic variation of run time of the tasks.

The load-balancing requirement driven directly by the SDP capital and power budgets, i.e., load-balancing is to be employed to maximise processing capability while keeping to power, capital budgets and a reasonable risk profile.

3.25 Load-balancing between observations

SKA observations are expected to have a wide range of computational requirements, varying by **orders of magnitude** between the most and least demanding. Furthermore, some stages of processing *within* an observation are more difficult to parallelise.

Load balancing between independent observations may be able to effectively smooth these different demands. Same criterion as Section 3.24 applies.

To load balance Z observations, the visibility buffer must be able to keep their observed data and therefore it needs to be of larger size:

$$M_{\text{buf}} \sim (1+Z)25\,\text{PetaByte} \tag{14}$$

4 Detailed questions

4.1 Energy use by the interconnect system

What is the predicted energy use by the interconnect system:

1. Expected static power usage per link/switch?
2. Dynamic power to transfer 1 bit

Should minimisation of interconnect traffic be a factor in power minimisation?

E.g. Zeller (2014) claims remote vs local DRAM uses 100 times more energy. Local DRAM is shown as (16 nJ per bit), so remote is $1.6\,\mu$J per bit, $\sim 10\mu$J per byte so 5 GigaByte link saturated with RDMA would be using 64kW (!) $\implies$ **does not make sense.**

What is the NVlik power usage per bit?

4.2 Reliability model

Should reliability be modelled as failure rate per compute unit, per node, per operation, per memory capacity? I.e., are 4000 nodes significantly more reliable than 40000 nodes with same FLOP throughput?

4.3 Multi compute-unit tightly-coupled kernels

E.g., tightly coupled FFTs and gridding which crosses multiple compute units? Is this worth it ? Will the programming models support it efficiently.

E.g. there are FFT implementations crossing multiple GPUs: what is the performance relative to single unit performance and what are energy costs?

4.4 Energy use sin/cos/sqrt functions

Radio astronomy fundamentally are based on sin/cos and sqrt functions with FFT/gridding used for efficiency. By doing more sin/cos computations it is possible to reduce the size of FFTs, potentially making them fit onto on-chip memory. A trade-off analysis would need as input the energy use per sin/cos instruction?

4.5 Latency for read operations from solid state storage

Read latency has significant impact on the software design for SDP. What is the expected read latency for solid state storage, bot locally and remotely?

References

Lucas, R. et al.: Top Ten Exascale Research Challenges, Tech. rep., DOE ESCAC Subcommittee Report, http://science.energy.gov/ /media/ascr/ascac/pdf/meetings/20140210/Top10reportFEB14.pdf, 2014.

Nikolic, B.: Estimating SDP Computational Efficiency, Tech. rep., SKA SDP Consortium, 2016a.

Nikolic, B.: Limits on the degree of faceting, Tech. rep., SKA SDP Consortium, https://confluence.ska-sdp.org/download/attachments/185794797/sdp-note-0008-facetinglimit.pdf, 2016b.

Zeller, C.: NVIDIAS VISION FOR EXASCALE, Tech. rep., Teratec Forum, http://www.teratec.eu/library/pdf/forum/2014/Presentations/SP04_C_Zeller_NVIDIA_Forum_Teratec_2014.pdf, 2014.