SDP Memo 54: Compute Node Pipeline Efficiency Assessment Framework

Document number	SDP Memo 54
Document Type	MEMO
Revision	1.1
Author	TN Chan, Anna Brown & Andrew Ensor
Release Date	2018-09-03
Document Classification	Unrestricted

Lead Author	Designation	Affiliation
TN Chan	System Architect	New Zealand Alliance / Compucon New Zealand

Signature & Date:	[illegible]	2018-09-03

SDP Memo Disclaimer

The SDP memos are designed to allow the quick recording of investigations and research done by members of the SDP. They are also designed to raise questions about parts of the SDP design or SDP process. The contents of a memo may be the opinion of the author, not the whole of the SDP.

Table of Contents

List of Abbreviations

A	An application or algorithm (small letter for subscript)
C	CPU (small letter for subscript)
CE	Compute Efficiency
CIP	Continuum Imaging Pipeline (small letter for subscript)
CL	Computation Load
FFT	Fast Fourier Transform
GB/s	Giga Bytes per second
GFLOPs	Giga Floating Point Operations per second
GPU	Graphic Processing Unit
ML	Memory (Transfer) Load
N	Number of Compute Devices (not Node)
OI	Operational Intensity
PE	Programming Efficiency
R	Ridge Point (small letter for subscript)
RE	Roofline Efficiency
SDP	Science Data Processor (small letter for subscript)
SIMD	Single Instruction Multiple Data
T	Run Time
W	Time Weighted Factor

1.0 Introduction

This document proposes a model for estimating the compute efficiency of a science pipeline in a compute node equipped with one or more hardware accelerators such as GPU. This information will support SDP system size estimation with a higher accuracy than achievable with the compute efficiency of a single algorithm.

- The model is based on the Roofline Model [RD1] published in April 2009 by the Association of Computing Machinery in its printed edition of Communications. It is applied to the following applications and hardware.

- Any SDP science pipeline consisting of one or more applications that can be executed in parallel to reduce the overall run time of the pipeline. An example is the Continuum Imaging Pipeline which consists of Gridding, FFT, and Multiple Scale Multiple Frequency Synthesis (MSMFS) in Deconvolution as the 3 major parallelisable algorithms.

- Standard x86 Accelerator model for commodity-off-the-shelf General Purpose Graphics Processing Unit (GPU).

The model has not considered scheduling of pipeline components other than assuming that they all run in the same compute node and the same GPGPU. This model is based on the report of an investigation TSK-2395 [RD2] and can be seen as a condensed version of the report.

2.0 References

2.1 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.

[AD1] SKA-TEL-SDP-0000040 Parametric models of SDP compute requirements

[AD2] Algorithm Reference Library http://www.mrao.cam.ac.uk/projects/jenkins/algorithm-reference-library/docs/build/html/ARL_background.html

[AD3] SKA-TEL-SDP-0000038 SDP System Sizing

2.2 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.

[RD1] Roofline Model published in April 2009 Volume 52 No.4 of the Communications of the ACM, Roofline: an insightful visual performance model for multicore architectures

[RD2] TSK-2395 Design Investigation Report: Compute Node SDP Sizing Estimation, https://confluence.ska-sdp.org/display/WBS/Compute+Node+SDP+Sizing+Estimation

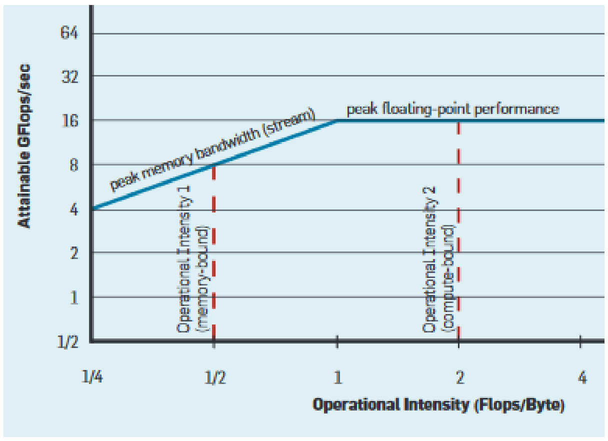
[RD3] TSK-1808 Design Investigation Report: Imaging Pipeline Computational Profile, https://confluence.ska-sdp.org/display/WBS/Compute+Node+Computational+Profile

[RD4] TSK-1916 Design Investigation Report: ARL Test CIP Benchmarking with GPU, https://confluence.ska-sdp.org/display/WBS/ARL+Test+CIP+Benchmarking+with+GPU

[RD5] TSK-2105 Design Investigation Report: https://confluence.ska-sdp.org/display/WBS/Compute+Node+Hardware+for+Domain+Science+Products

3.0 Science Pipeline Compute Node Efficiency Modelling

- The Roofline Model expresses graphically the attainable floating point operational capability per second (herewith abbreviated as GFLOPs where G is 10^9) of a computing hardware device for various values of the operational intensity. For the convenience of referencing, an image taken from [RD1] is reproduced herewith.



- The operational intensity (OI) was defined as the ratio of GFLOPs and attainable memory bandwidth in Byte per second (herewith abbreviated as GB/s where G is 10^9). GFLOPs is the vertical or Y-axis and OI is the horizontal or X-axis of a 2D graph. A specific hardware device would have a slanted line for the low end of the OI and a horizontal line for the higher end of the OI and they intersect at the ridge point resembling the roofline of a house. The slanted line is equivalent to the Attainable GB/s of the hardware device.

  (OI) = GFLOPs divided by GB/s .... (1)

  Attainable GFLOPs = min [(Peak GFLOPs), (Attainable GB/s * (OI))] .... (2)

  Where
  - The Peak GFLOPs is the same as the theoretical peak that can be calculated from published hardware specifications.
  - The Attainable GB/s is obtained by running a series of micro-benchmarking applications that the authors of the Roofline Model paper developed on the device. This benchmarked peak memory bandwidth is a lot lower than the theoretical peak memory bandwidth that can be calculated from published hardware specifications. However, the paper did not disclose the magnitude of the difference.
  - OI is as defined by Equation (1)
  - Symbol * is multiplication (not convolution)

- A computing hardware device has one Roofline which remains the same irrespective of the software code running in it. In order to locate where a specific software program would sit on the graph, the OI of the software program A $(OI)_a$ has to be determined first. The OI is determined by dividing the total GFLOP of the program by the total amount of data read and write in GB of memory required by the program. This is expressed in Equation (1). If the $(OI)_a$ is higher than the $(OI)_r$ coordinate of the ridge point, the horizontal line is the ceiling of computational resource that is available to the software. If $(OI)_a$ is lower than $(OI)_r$, the slanted line is the ceiling. This indicates the importance of the ridge point of the device to the performance of software applications.

- The ridge point indicates the level of difficulty for programmers to achieve the maximum GFLOPs or GB/s from the hardware for a specific software program. The level of difficulty is higher if the ridge point is located away from the graph origin towards the right hand side.

3.1 Applying Roofline Model

- The OI of each algorithm is given by the ratio of compute load (CL) in GFLOP and memory read/write load (ML) in GB such as given by the SDP Parametric Model [AD1] or SDP Algorithm Reference Library (ARL) [AD2]. That is, Algorithm OI values do not pertain to any computer hardware devices and are not obtained from benchmarking.

  $(OI) = CL / ML$ as given by Equation (1)

- A pipeline such as the Test Continuum Imaging Pipeline examined in previous SDP investigations [RD3], [RD4], [RD5] consists of 5 major algorithms in series- Gridding, iFFT, Deconvolution, FFT, and De-gridding. The challenge is to convert the 5 (OI) to a composite (OI) for the pipeline. The following is such an attempt.

  Computational Capability CC in GFLOPs = CL / T ….. (3)
  Memory Bandwidth in GB/s = ML / T ….. (4)

  Where time T is measured given a specific dataset and a specific hardware device.

- The Composite OI of the pipeline can be obtained by assuming that all 5 algorithms exist as one entity with a single computational pattern. As far as the Test Continuum Imaging Pipeline (CIP) is concerned, this assumption is not valid as the computational pattern varies over time for the algorithms. The equation below is therefore for reference only.

  $(OI)cip = \Sigma\, (CL)i / \Sigma\, (ML)i$ ….. (5)
  Where the suffix i stands for each major algorithm component of the pipeline

  - A more realistic approach is to use benchmarking test results as a basis. Benchmark the 5 algorithms and calculate the proportion of the total pipeline time taken by one algorithm to give Wi as the time-weighted factor.

    $Wi = Ti / \Sigma\, Ti$ where $i = 1$ to 5 ….. (6)
    $(OI)cip = \Sigma\, [\, Wi * (OI)i \,]$ ….. (7)
    Where (OI)cip is the Composite OI of the pipeline. This figure can be used to locate the OI of the pipeline on the Roofline diagram of the hardware device used for benchmarking or installation.

- Equation (7) ignores the fact that no matter how far an (OI) is on the RHS of the ridge point, it is effectively located at the ridge point as far as computation ceiling is concerned. As such all components that sit under the flat roof should be considered to be sitting directly under the ridge point as expressed in (8) below.

$(OI)_{cip} = \Sigma [W_i * (OI)_i] \text{ if } (OI)_i < (OI)_r + \Sigma [W_i * (OI)_r] \text{ if } (OI)_i => (OI)_r$ ..... (8)

- Equation (8) allows the attainable peak compute capability to be obtained from the Roofline diagram of a specific hardware device.

3.2 Compute Efficiency vs Programming Efficiency

- The location of the Composite Pipeline (OI) on the Roofline diagram exposes two optimisation opportunities- horizontal and vertical.

- Horizontal: If it sits under the slanted part of the roofline, programming efforts should attempt to shift the (OI) towards the RHS and preferably past the ridge point to attain 100% of Roofline Efficiency. Otherwise, the Roofline Efficiency (RE) would be less than 100% and is expressed by:

  $RE = [\text{Attainable GB/s} * (OI)_{cip}] / \text{Peak GFLOPs}$ ..... (9)

- Vertical: Irrespective of where the algorithm OI sits on the Roofline diagram, programming efforts are needed to push up the location of the algorithm towards the Roofline. The Programming Efficiency (PE) would be 100% if the algorithm touches the Roofline, or else PE would be less than 100%.

- In summary, these 2 efficiencies make up the Compute Efficiency (CE) of an algorithm or a pipeline

  $CE = RE * PE$
  Where * is multiply (not convolution)

- There are 6 or more types of optimization programming actions on an algorithm. The first 2 actions move the OI of an algorithm to a more favourable location. The next 4 actions push up the location of the algorithm towards the roofline.

  - Increase the computational load without increasing the memory transfer load
  - Decrease the memory transfer load without decreasing the computational load
  - Improve instruction level parallelism and apply SIMD
  - Create multiply-accumulate (MAC) instructions to utilise hardware provisions
  - Ensure memory affinity if the compute node consists of multiple CPU chips (e.g. Dual Socket Intel Xeon or AMD EPYC CPU which is a package of multiple chips)- this is to ensure that a compute process does not need to cross inter-chip links to access data
  - Apply software pre-fetching of data instead of waiting for an application to initiate memory transfer

3.3 SDP Size Estimation

- The number of compute devices (N) can be easily obtained from dividing the total compute load (CL) as estimated by the SDP Parametric Model [AD1] by the Peak compute capacity (CC) of a hardware device realisable by the Compute Efficiency (CE).

$$N = CL / [CC * CE] \qquad \text{..... (10)}$$

- Equation (10) provides a GPU view only assuming the compute device is a GPU. As far as the entire pipeline is concerned, the CPU provides a unified view. For a specific science pipeline such as CIP, the CPU executes the host code in Tc seconds and issues each parallel kernel to be executed in the GPU in Ti seconds in sequence where i stands for the number of parallelisable algorithms.

$$Tcip = Tc + \Sigma\ Ti \qquad \text{..... (11)}$$

- SDP is a collection of science pipelines. The total GFLOPs could be established by adding the time weighted GFLOPs of individual pipelines (Wj) for one observation period (Tsdp). Wj is known from the High Priority Science Objective allocations [AD3]. Assume Tj runs in series and uses the full SDP hardware resources,

$$Tsdp = \Sigma\ (Wj * Tj) \qquad \text{..... (12)}$$

- Tsdp is the observation period and it should be discounted to 80% or 90% (figures are arbitrary) of the observation period leaving a legroom for other computer applications outside of science pipelines such as SDP task scheduler and shared services.

4.0 List of Assumptions Made

a) The concepts expressed by the Roofline model were faithfully observed

b) The standard x86 Acceleration model was applied. The CPU-only scenario has not been investigated due to the belief that the low computational capability of CPU will not satisfy SDP applications economically and perhaps technically due to scaling issues.

c) A single compute node was used for executing the entire pipeline

d) Source code of the pipeline to be measured was taken from ARL.

e) Input data shall be taken from a simulator which is considered to be representative of the SDP data for a correct prediction of the operational intensity and runtime of each algorithm. A data shape and data set size shall be determined for this modelling process. The data shape and data set size would be limited by the CPU main memory and GPU global memory available in the compute node being tested.

f) Algorithms of the science pipeline shall be identified first and separated into 2 categories of sequential and parallelisable respectively. The pipeline shall be executed in a CPU compute node and the portion of runtime for the sequential code shall be measured. The runtime is Tc in Equation 11.

g) The parallelizable code of each major algorithm of the science pipeline shall be converted with pyCUDA for executing in a GPU. The location of the operational intensity of each on the Roofline Diagram of the GPU device shall be revealed by real life benchmarking and they are (OI)i respectively in Equation 8. The runtime is Ti in Equation 6 and 11. (Note: The OI obtained from a parametric model places the algorithm on the X-axis but not yet the Y-axis)

h) Since SDP consists of a number of pipelines, the total SDP runtime is the sum of individual pipeline runtimes weighed by the time allocation as decided by HPSO Allocations. The weights are wj in Equation 12.

5.0 Post Script Discussions for Revision 1.1

5.1 Bojan Nikolic

a) Comment: It seems it does not refer to memo #10. Could you add how this approach compares and contrast to what was written in memo #10?

Response: Memo #10 was known as Memo 86C at the time of drafting the report for TSK-2395 which was the predecessor of Memo 54. See https://confluence.ska-sdp.org/display/WBS/Compute+Node+SDP+Sizing+Estimation. It did refer to Memo 86C among various other documents of a similar context. Memo 54 was a condensation of TSK-2395 and it skipped a lot of details. Not mentioning Memo 86C and several other references in Memo 54 was part of the condensation process.

Memo 10 established an operational intensity and compute efficiency figure for SDP with 2 steps that were not used in Memo 54 or different in approach to Memo 54.

- It obtained the memory transfer rate for establishing the operational intensity (OI) of an algorithm by measurement or inference.
- It obtained the SDP compute efficiency by using the aggregate compute load and the measured (or inferred) memory bandwidth.

Memo 54 established a theoretical OI of an algorithm (not aggregate of SDP) from the parametric model by dividing the compute load by the data transfer load. The OI was then placed on the x-axis of the Roofline Diagram of a specific hardware device. That is, the OI was theoretical and totally based on the parametric model and not on any hardware device at all. Where on the y-axis did the algorithm sit was not known. The actual location of the algorithm on the Roofline Diagram (x, y) if known will indicate its compute efficiency and this locating step depends on how the application was programmed and how it made use of the hardware resources of the specified hardware device. This memo continued to suggest a way for obtaining a composite OI of a pipeline, and provided further ideas for unifying with the actual wall clock runtime incurred by the CPU for various pipelines.

The 2 memos are therefore different in terms of approach.

b) Comment: The statement that Algorithm OI is independent of computer hardware and not obtained from benchmarking is not correct I think. Most obviously cache size (versus problem size) critically affects the OI. But also much subtler things like h/w prefetch, speculative execution and others impact OI, and also interact with different implementation of same algorithm in a complex way. So, I would suggest benchmarking is the only realistic way of infer OI.

Response: Memo 54 started with the parametric model if the model had supplied the compute load and data load for each major algorithm. This was only a starting point for the system size estimation exercise. It is totally agreeable that the final OI and compute efficiency have to be obtained by benchmark testing in specific hardware devices to be realistic. We talk about estimation versus prototyping here.

c) Comment: Also, there is a small point but one that seems to come up often: improving OI by increasing the computational load is pointless since it does nothing to improve the time to solution – yes efficiency is higher, but there is correspondingly more work to do cancelling each other.

> Response: There is a different way of seeing the same suggestion as it was first publicly brought out by John Gustafson on Gene Amdahl's expression of the maximum speed up attainable subject to the portion of code that is not parallelizable. www.johngustafson.net/pubs/pub13/amdahl.pdf. By increasing the computational load, one would expect more productivity from the load. This could be a transfer of a computation parcel from Task A to Task B such that Task A benefits at no cost to Task B for example.

5.2 Peter Wortmann

a) Comment: Another point is that the idea of a "composite OI" seems somewhat questionable. If we have a pipeline that has stages with very different OIs (like FFT and gridding) that makes "averaging" the OI misleading. After all, it would tell us to pay too much attention to the "centre" of the roofline diagram and seeking a sharp "ridge point", when in actuality we are dealing with a complex trade-off between the ceiling and the situation at lower OIs. The only way averaging would make sense is if we assume that we could mix the execution of different stages at a low level a la hyper-threading, but that possibility is too situational and fragile to be considered seriously.

> Response: The comment is totally agreeable and Memo 54 suggested a reasonable compromise for SDP size estimation purposes assuming a composite OI is desirable for system size estimation. The suggestion is based on the theory that the level of certainty of estimation based on the centre OI would be higher if the spread of data is smaller. At first thought, this theory would rule out any attempts of averaging the OI of FFT and Gridding since they are indeed quite different. However, Memo 54 reduced the separate distance between their OI substantially without losing substance by saying that no matter how high the Gridding OI is, it is no farther away than the Ridge Point as far as the access to the flat compute ceiling is concerned. In case FFT was under the slanted part of the roofline and Gridding was at the ridge point, a time-weighted average would be on the slanted part but closer to the ridge point than FFT. This centre point is representative of the small OI spread for estimation purposes.

b) Comment: It would seem that one would have to examine how important different algorithms with characteristic OIs are, then derive a composite performance score from that? Things are even more interesting due to the fact that we will likely have compute capabilities with different characteristics available. I think this is the kind of detail we have to go to here to meaningfully evaluate a SDP hardware configuration or pipeline.

> Response: Memo 54 made the assumption that a parallelizable algorithm taking the longest runtime is the most important, and suggested to obtain the relative runtimes of various algorithms by testing them in the same hardware device in order to produce the weights needed for estimating the composite OI. The idea of looking into compute capabilities with different characteristics is welcoming. Memo 56 (a different memo) suggested that data intensive algorithms were hostile to scaling,

and this suggestion can be further examined together with compute efficiency in the next investigation.

c) Comment: Are there plans for actually applying the model? Even if I have some doubts about the method, some results would be interesting to look at.

Some interpretation of the prototyping results obtained by the Oxford Vertical Prototyping team on Gridding with this memo was floated recently for internal discussion. The interpretation was supportive of the model but not conclusive. Some further work in the area is desirable when more prototyping data become available.

End of Memo