Subblock Posturing in OCR Applications

Brian R. Nickerson

Intel Corporation

# Legal Boilerplate

**Acknowledgment:** This material is based upon work supported by the Department of Energy [Office of Science] under Award Number DE-SC0008717.

**Disclaimer:** This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

**Cheese:** “I like the posture, but not the yoga.” ― Roman Payne

# The Big Idea

**Posture (verb): To position, especially strategically. (dictionary.com)**

Key concept: a “Subblock” is a subset portion of a larger datablock (which I subsequently refer to as the “Backing-store” datablock). Presently, it is a “pencil” of a larger 1D (or higher order) structure, a subset rectangle of a 2D (ibid) structure, a subset cuboid of a 3D (ibid) structure, and so on into higher dimensions. The larger structure may have been allocated by one EDT, and then split into subblocks to be initialized by one or more other EDTs; and after it has been initialized, it might be split among one or more consumers.

Key concept: The verb “Posture” is intended to convey a passive concept, as contrasted with the verb “Migrate” which is imperative. By way of analogy, a military commander is “posturing” resources both when she decides to deploy them into action, and when she decides to hold them in reserve.

If an upstream EDT tells OCR to “migrate” a subblock to a downstream EDT, it is essentially decreeing that the subblock must be extracted from the larger datablock, packed into a new datablock (with multi-dimensional shape spans reduced accordingly), and passed to the downstream EDT. OCR still gets to decide where to put the subblock, but it has no choice about whether or not this was a good idea (as opposed to the downstream EDT using the subblock from out of its original part of the full datablock). The downstream EDT is inherently dependent on the shape of the migrated datablock.

Contrastively, a directive that OCR should “posture” a subblock to a downstream EDT means that the upstream EDT is committing the downstream EDT to a contract, that the latter must adapt to a decision that OCR will make, at its own discretion, as to whether to actually migrate the subblock, or leave it in the larger datablock. Implicit in this is that the downstream EDT will need to adapt to the shape of the subblock, which may or may not be packed.

# Current State of Affairs: To Migrate or Not to Migrate, That is the Question

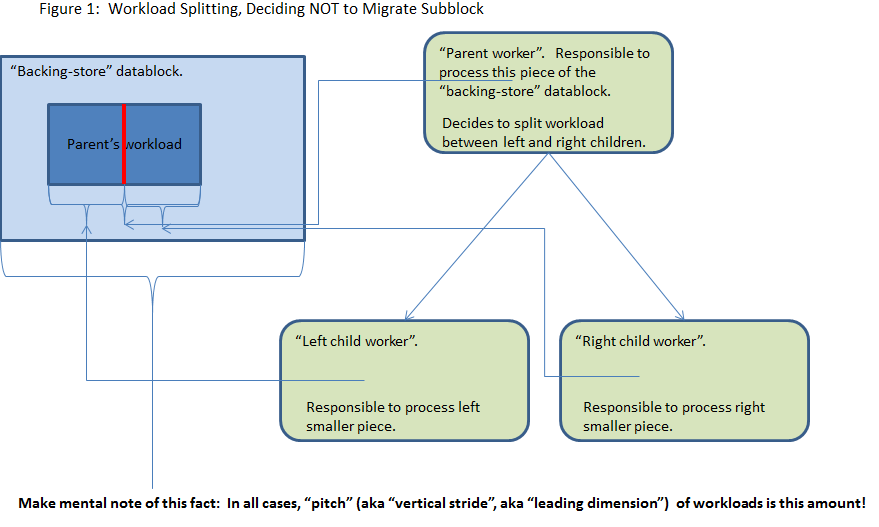
A parent worker EDT, intent upon splitting a workload between two or more children, must make the executive decision as to whether to migrate pieces (subblocks) of the overall workload to new, tighter-packed datablocks and feed those to the children; or else tell the children to mine the larger aggregate datablock for the portions they are responsible to process.

But expecting the parent to know whether it is “wise” to migrate the pieces imposes upon that parent a context sensitivity that is totally unjustified: the parent needs to know whether there is suitable storage available for the pieces. (“Suitable” generally means “closer to the agent(s) that will use the subblocks”. This is NOT guaranteed, and in pathological cases, the subblocks might end up in storage that is actually farther away from the computing agent(s) than the original aggregate datablock.) It is bad enough to expect the parent EDT to have knowledge of the static characteristics of the memory architecture (how much memory exists at various levels, and what are their access costs in terms of cycles and energy). It is even more unrealistic to expect the parent EDT to know how much of those resources are actually going to be available dynamically, when the child(ren) EDTs run, as opposed to being in use by other EDTs and/or by OCR itself. Also, bear in mind that in the fullness of time, a real exascale system is likely be comprised of components that are not exactly identically resourced.

So if the parent decides to migrate, pathological placement might result; and excessive, unnecessary block movement is also likely. But if the parent decides NOT to migrate, there is also a very good chance that the aggregate is farther away from the child(ren) than the migrated subblocks would have been, so this, too, wastes time and energy.

Moreover, in the current paradigm, there is no means to tell OCR that two or more children that want to write to subblocks of the larger aggregate are “well-behaved”, i.e. that the will be writing to disjoint subblocks. OCR presently assumes the writers might write anywhere in the aggregate, so it is forced to serialize access. This is a major performance problem for applications; ***and it is elegantly solved by Posturing!***

Figures 1 and 2 depict the To-Migrate and Not-To-Migrate limited choices presently available, sans posturing.



# Subblock Posturing – Tenets

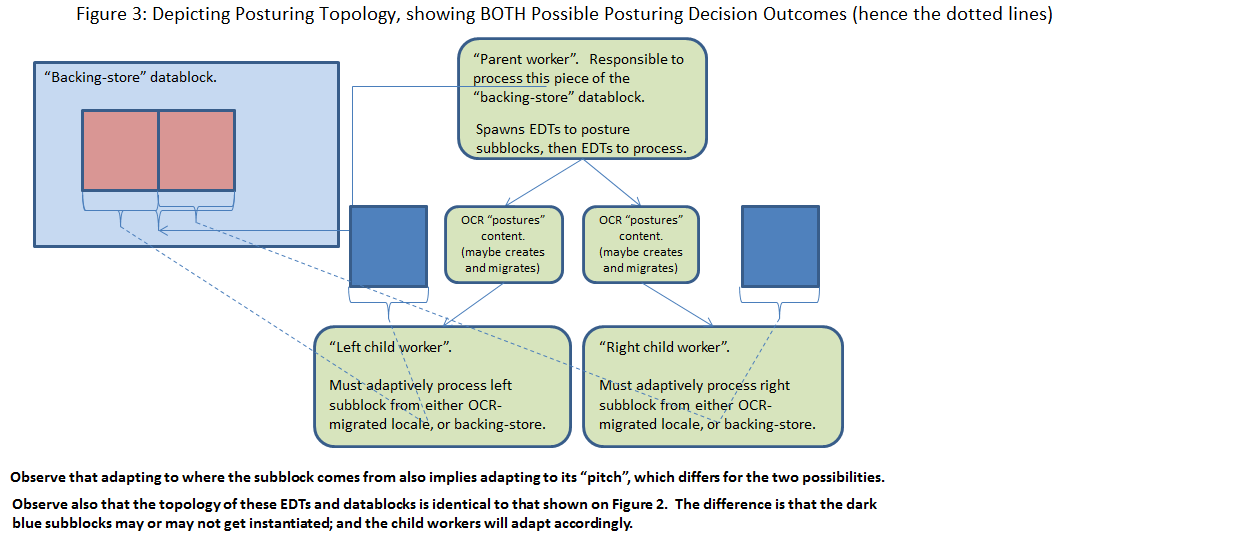
* Important power, energy, and speed economies come from wisely boosting workload subblocks to memory resources closer to the computing agents that will utilize them.
  + *When subblocks are NOT migrated but should be, access economies are wasted.*
  + *When subblocks are migrated that should not be, excessive copying is likely, and pathological case is that migrated subblocks could end up farther away from the computing agents than their underlying backing store!*
* Application should be “context insensitive” to what memory resources exist, moreso their relative performance profiles, and much moreso their current availability. So application should not have to guess if it should demand data migration.
* OCR should have the freedom to move subblocks closer to consuming computing agents, or NOT – and application should “just work” either way.
* OCR should be able to make a late decision on whether the subblocks will or will NOT be migrated. Basis will be where consuming EDT(s) get scheduled, and the very dynamic nature of resource availability at that time.
* Regardless of whether OCR decides to migrate or not, it should take full advantage of the assurance that EDTs that will write to disjoint subblocks can be scheduled to run simultanesously.

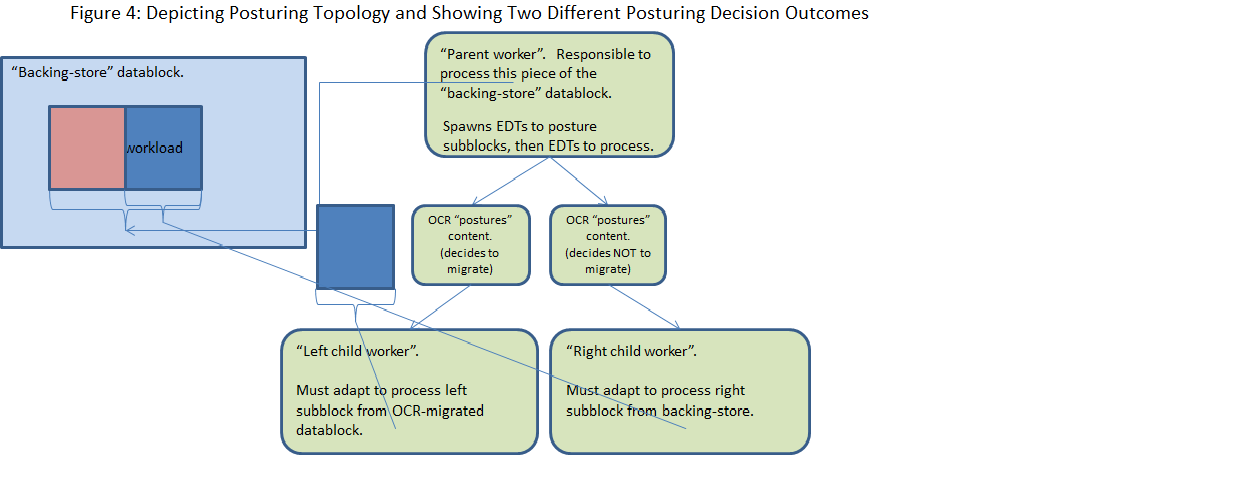
# Subblock Posturing – Mechanism

The author has no strong opinion on what the exact mechanism should be for negotiating with OCR to do a posturing activity. He has, however, implemented a Proof of Concept prototype utilizing just a normal EDT syntax, and has written a library of EDTs that perform input and output posturing activities. In Figures 3 and 4, the EDTs entitled “OCR postures content” are the input-oriented parts of that library.

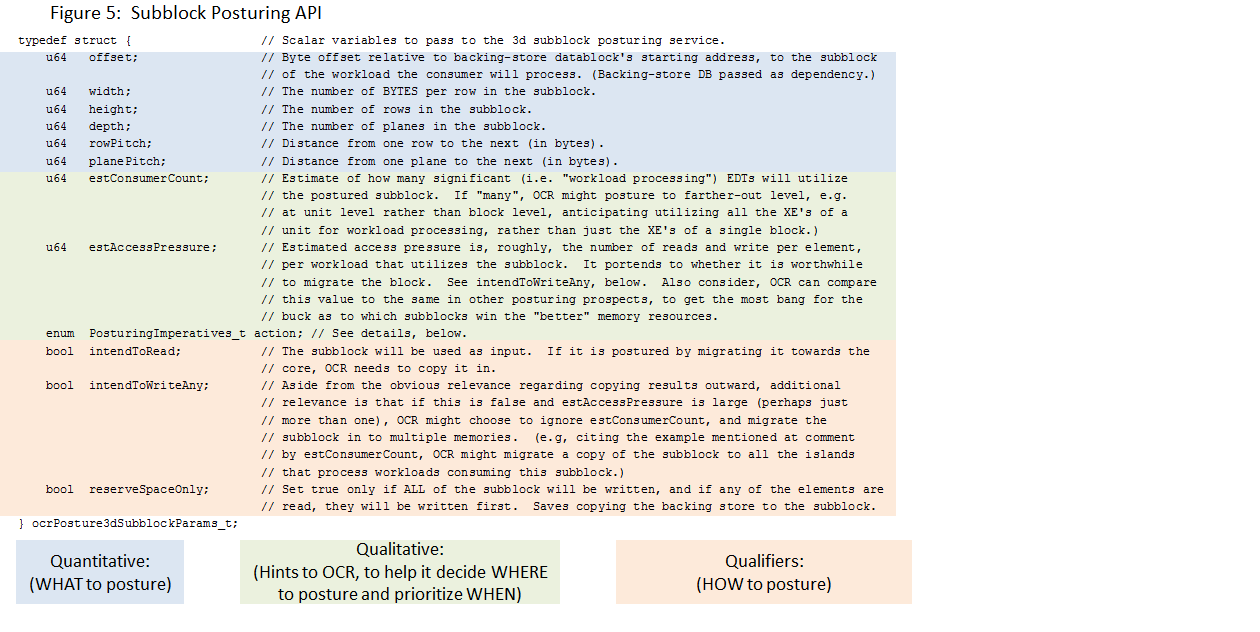
This implementation has allowed me to unit test the concept, which I have done with GEMM. The unit test is not capable of making an “intelligent” decision about whether or not to migrate – that will take the full system awareness that is the proper purview of OCR – but it attains testing coverage by making a pseudo-random decision of whether or not to migrate.

As suggested subsequently, this EDT-like mechanism might be a good way to go longer term. The EDTs that I use could perhaps become OCR primitives. Looking like EDT instantiations (ocrEdtCreate) with their parameters and dependencies, they might be intercepted and handled specially by OCR. That is for the OCR architects to decide.





# Posturing API (at least, as currently implemented for Proof of Concept)



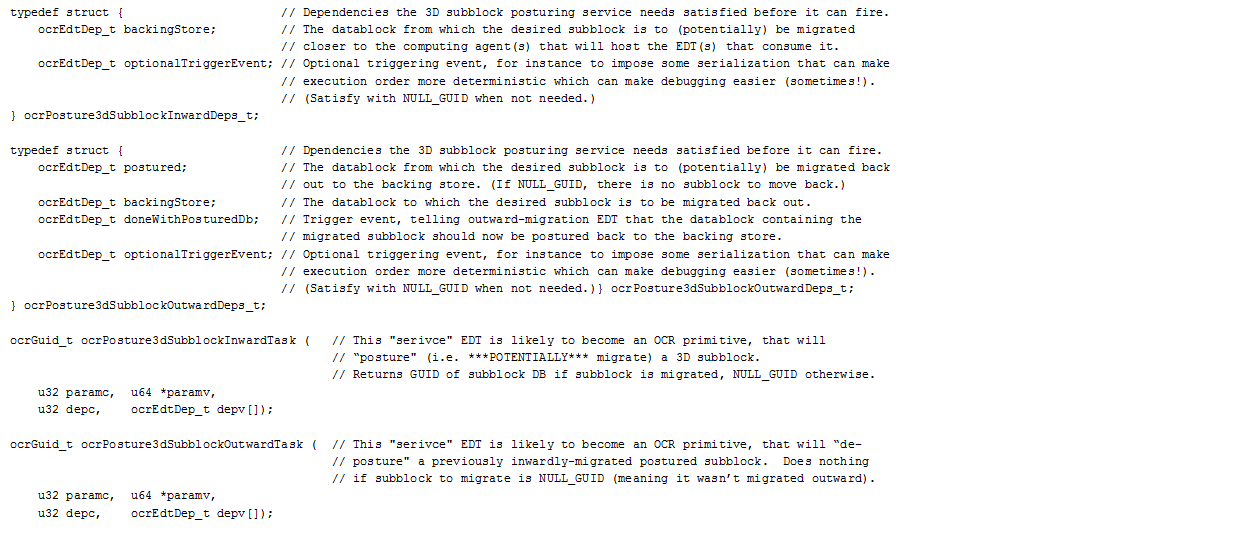


Figure 5 mentions the PosturingImperatives\_t element. The idea here is to provide these values, for these reasons:

* OcrDecides: OCR is entirely free to decide whether or not go posture, and should make the decision “intelligently”, adapting to available resources.
* ForceMigration: The “intelligent” decision being made by OCR is overridden with an imperative that the migration MUST be performed. This functionality is essentially the same as the existing functionality of Figure 2, but it can be extremely convenient. For example, suppose one is doing a GEMM of a very large pair of matrices. Suppose the overall workload is being split binarily, and the splitting will continue until reaching leaf EDTs that will have a suitable “bite-sized” workload. In the descent to the leaf EDTs, posturing can be done with the anticipation that as left and right, or up and down subblock workloads get small enough to fit into smaller, closer-to-the-computing-agents memories that will process them, OCR will indeed make these posturing operations migrate the data. However, if we want to FORCE the final leaf EDTs to work on datablocks that contain only their workload (as opposed to having to mine it from a larger aggregate “backing-store”) rather than have to write the logic in the parent-of-the-leaf EDT to use the force-migration paradigm (the existing facility, depicted in Figure 2), it is more convenient to simply use the same API as for posturing done at all the other levels, except to throw this particular PosturingImperatives\_t element value.
* ForceNonMigration: This option overrides OCR, forcing it NOT to migrate, which is akin to the current mechanism as depicted in Figure 1, with the parent forcing the child to mine the original datablock for the subblock it needs to operate upon. However, it has the very valuable advantage of endowing OCR with the knowledge and assurance that disjoint subblocks are indeed disjoint, and so can be scheduled to be operated upon simultaneously. Without this facility (or something else like it), OCR cannot know the disjoint relationship of subblocks marked for writing, but instead must assume that the entire backing-store datablock “belongs” to each EDT in turn, forcing their write access to serialize. Even read-only cases potentially benefit, because only the subblock needs to be moved closer to a consuming EDT, rather than the full backing-store datablock (if it is even possible).
* During unit testing, the user can also temporarily impose the ForceMigration and then the ForceNonMigratino options to make sure that both behaviors are debugged, before switching back to OcrDecides for production runs.
* Note that this element carries more imperative than a mere hint. Hints in OCR can be entirely ignored, whereas ignoring some of these imperatives could lead to pathology. Since the whole Posturing API is more complicated than could be embodied in a mere hint upon existing APIs, it is best to NOT think of this element (and the other boolean “qualifiers”) as “hints”.

These are the actual EDT top-level functions:

* ocrPosture3dSubblockInwardTask
  + Presently looks like any other EDT, but …
  + Destined to become an OCR primitive that …
  + Uses scheduling “magic” to decide when a “postured” datablock should or should NOT be migrated.
  + Instantiated as a “Finish” EDT.
  + Returns GUID of either the backing-store datablock (if migration NOT done) or of the migrated subblock.
* ocrPosture3dSubblockOutwardTask
  + Presently looks like any other EDT, but …
  + Destined to become an OCR primitive as well, and …
  + Its job (not depicted in earlier animations) is to return modified subblocks to backing-store datablocks,
  + And free up the subblock datablocks.
* When OCR decides to migrate, it will also configure and fire DMAs to initialize input subblocks, and the …OutwardTask will do the DMAs that will copy them back out.

# Using the API from Application Code:

* Duties of the User’s parent EDT:
  + Receives as an input dependence the larger datablock, subsequently referred to as the “backing-store”.
  + Instantiates ocrPosture3dSubblockInwardTask as a “Finish” EDT.
  + Instantiates its child (that will work on a subblock of the “backing-store” datablock) as a “Finish” EDT.
  + Adds output of ocrPosture3dSubblockInwardTask as a dependence to its child EDT.
    - That will be the guid of the backing-store if the posturing EDT chose NOT to migrate the subblock.
    - Otherwise it will be the guid of the subblock’s datablock, created (and initialized) by the posturing EDT.
  + Also provides the guid of the backing-store to the child as a parameter.
  + Also provides as parameters the shape and offset information of the subblock as situated in the backing-store; and the shape information of the subblock as situated in the migrated subblock.
  + Instantiates ocrPosture3dSubblockOutwardTask, plumbs the subblock datablock, the backing-store datablock, and the child’s done event as dependences to it; and the shape information as parameters, as well as a flag indicating whether the subblock might have been written by the child.
* Duties of the User’s child EDT – simple adaptation to OCR’s posturing decision:
  + If the guid received as a dependence for the input subblock datablock is equal to the guid received as a parameter for the backing-store datablock, it means OCR decided NOT to migrate the subblock. The user code needs to access the subblock directly from the backing-store, utilizing those parameters in the input that describe that representation’s offset, height, width, and depth information.
  + Otherwise, if the guids are NOT equal, it means that the guid received as a dependency is that of the migrated subblock. The user code needs to use the other parameters relating to the height, width, and depth information for the migrated block’s representation. In this case, the offset is zero.
  + If that’s not crystal clear, refer back to Figure 4.

# Proof of Concept:

* DGEMM already demonstrates this facility.
  + Works great!
  + The tasks destined to become OCR primitives are stub code that imitate what OCR would do – except that the “magic” decisions of whether or not to migrate the subblocks is decided on a pseudo-random basis.

# Ponderables:

* It might prove desirable to allow subblocks to be described by three different shape characteristics: the storage size of the subblock (if it is indeed postured), how much (and what part) of that shape is to be brought in as input, and how much (and what part) of that shape is to be written back to the backing-store as output. There is tedium in specifying all this added detail, and probably some potential for pathology that must be characterized and precluded. But it might be helpful for dealing with stencil halos and the like.
* While the API described in Figure 5 anticipates 3D datablocks and subblocks, and can be utilized for 1D and 2D, it is likely that higher dimensionalities will be needed for some applications. There is no reason the concepts shouldn’t be generalized for such cases.
* In the fullness of time, it might be desirable to accommodate other shapes of subblocks than pencil/rectangular/cuboid, etc. For instance, when dealing with triangular matrices, it is likely that both the backing-store datablock and various subblocks will utilize rectangular storage constructs, but the actual reading and writing migrations would often be more efficiently done as just sub-triangle transfers rather than sub-rectangles. It is plausible that we might want to implement a few migration primitives, but allow a programmer to write a more detailed migration prescription in cases where it would yield better performance.