

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

REALTEK SEMICONDUCTOR CORP.,
Petitioner,

v.

ATI TECHNOLOGIES ULC,
Patent Owner.

IPR2023-00687
Patent 8,854,381 B2

Before JAMES P. CALVE, BRIAN J. McNAMARA, and KEVIN W. CHERRY,
Administrative Patent Judges.

CALVE, *Administrative Patent Judge.*

PRELIMINARY GUIDANCE
PATENT OWNER'S MOTION TO AMEND

I. INTRODUCTION

On October 25, 2023, we instituted trial as to claims 15–20 of U.S. Patent No. 8,854,381 B2 (“the ’381 patent,” Ex. 1001). Paper 7 (“Inst. Dec.”). After institution, Patent Owner filed a contingent Motion to Amend (Paper 13 (“Mot.”)) presenting proposed substitute claims 28 and 29 and requesting that “[i]f during this IPR proceeding, claim 16 is found unpatentable, . . . it be replaced with proposed substitute claim 28” and “[i]f claim 17 is found unpatentable, . . . it be replaced with proposed substitute claim 29.” Mot. 2; *id.* at 25. Patent Owner submitted a third Declaration of John Hart, Ph.D. (Ex. 2008) in support of the Motion. Petitioner filed an opposition on April 10, 2024. Paper 19 (“Opp.”). Petitioner submitted a third Declaration of Massoud Pedram, Ph.D. (Ex. 1014) in support of the Opposition.

In the Motion, Patent Owner requested that we provide preliminary guidance concerning the Motion in accordance with the Board’s pilot program concerning motion to amend practice and procedures. Mot. 2; *see also* Notice Regarding a New Pilot Program Concerning Motion to Amend Practice and Procedures in Trial Proceedings under the America Invents Act before the Patent Trial and Appeal Board, 84 Fed. Reg. 9,497 (Mar. 15, 2019) (providing a patent owner with the option to receive preliminary guidance from the Board on its motion to amend) (“Notice”). We have considered Patent Owner’s Motion and Petitioner’s Opposition.

In this Preliminary Guidance, we provide information indicating our initial, preliminary, non-binding views on whether Patent Owner has shown a reasonable likelihood that it has satisfied the statutory and regulatory requirements associated with filing a motion to amend in an *inter partes* review and whether Petitioner (or the record) establishes a reasonable likelihood that the substitute claims are

unpatentable. *See* Notice, 84 Fed. Reg. at 9,497 (“The preliminary guidance . . . provides preliminary, non-binding guidance from the Board to the parties about the [motion to amend]”); *see also* 35 U.S.C. § 316(d) (statutory requirements for a motion to amend); 37 C.F.R. § 42.121 (regulatory requirements and burdens for a motion to amend); *Lectrosonics, Inc. v Zaxcom, Inc.*, IPR2018-01129, Paper 15 (PTAB Feb. 25, 2019) (precedential) (providing information and guidance regarding motions to amend).

For purposes of this Preliminary Guidance, we focus on the proposed substitute claims, and specifically on the amendments proposed in the Motion. *See* Notice, 84 Fed. Reg. at 9,497. We do not address the patentability of any of the originally challenged claims. *Id.* Moreover, in formulating our preliminary views on the Motion and Opposition, we have not considered the parties’ other substantive papers on the underlying merits of Petitioner’s challenges. We emphasize that the views expressed in this Preliminary Guidance are subject to change upon consideration of the complete record, including any revision to the Motion filed by Patent Owner. Thus, this Preliminary Guidance is not binding on the Board when rendering a final written decision. *See id.* at 9,500.

II. PRELIMINARY GUIDANCE

A. Statutory and Regulatory Requirements

For the reasons discussed below, at this stage of the proceeding, and based on the current record, it appears that Patent Owner has shown a reasonable likelihood that it has satisfied the statutory and regulatory requirements associated with filing a motion to amend with respect to proposed substitute claim 29. However, it appears that Patent Owner has not shown a reasonable likelihood that it has satisfied these requirements for proposed substitute claim 28.

1. Reasonable Number of Substitute Claims

Does Patent Owner propose a reasonable number of substitute claims?
(35 U.S.C. § 316(d)(1)(B))

Yes. Patent Owner proposes no more than 1 substitute claim for each of challenged claims 16 and 17. Mot. 2–3, 25, App. A 1–2 (Claim Listing). Petitioner does not argue otherwise. *See generally* Opp.

2. Respond to Ground of Unpatentability

Does the Motion respond to a ground of unpatentability involved in the trial? (37 C.F.R. § 42.121(a)(2)(i))

Yes. Patent Owner responds to the grounds of unpatentability on which we instituted trial. Mot. 3–4, 12–25. Upon review of Patent Owner’s arguments, we agree that proposed substitute dependent claims 28 and 29 recite new limitations, and new combinations of limitations, that directly respond to the grounds of unpatentability involved in the trial. *See* Mot. 3–4, 12–25, App. A 1–2 (Claim Listing). Petitioner does not argue otherwise. *See generally* Opp.

3. Scope of Amended Claims

Does the amendment seek to enlarge the scope of the claims? (35 U.S.C. § 316(d)(3); 37 C.F.R. § 42.121(a)(2)(ii))

No.

Proposed substitute dependent claims 28 and 29 (i) retain all limitations of their original corresponding claims 16 and 17, (ii) include additional limitations as compared to claims 16 and 17, and (iii) depend from original (i.e., unchanged) claim 15. *See* Mot. App. A 1–2 (Claim Listing). Therefore, proposed substitute claims 28 and 29 do not enlarge the scope of the claims of the challenged patent. *See id.* at 4, App. A 1–2 (Claim Listing). Petitioner does not argue otherwise. *See generally* Opp.

4. New Matter

Does the amendment seek to add new subject matter? (35 U.S.C. § 316(d)(3); 37 C.F.R. § 42.121(a)(2)(ii))

Yes as to proposed substitute claim 28.

No as to proposed substitute claim 29.

Proposed Substitute Claim 28

On this record, Patent Owner does not appear to have sufficiently identified adequate written description support for proposed substitute claim 28. *See Lectrosonics*, Paper 15 at 7 (requiring Patent Owner to “set forth written description support in the originally filed disclosure of the subject patent for each proposed substitute claim”); *see also* 37 C.F.R. § 42.121(b)(1).

Proposed substitute claim 28 recites the added limitations of the shader core comprising “multiple processing elements,” “wherein each of the plurality of tasks is decomposable into two or more related threads,” and “wherein at least one processing element is configured to receive tasks from each engine.” Mot. App. A 1–2 (Claim Listing).

Patent Owner asserts that “multiple processing elements” is disclosed at least in paragraphs 30 and 50–54 and Figure 2, and “wherein each of the plurality of tasks is decomposable into two or more related threads” is disclosed at least in paragraphs 5–8 of U.S. Application No. 12/874,134 (“the ’134 application”),¹ which issued as the ’381 patent). Mot. 7–8 (citing Ex. 1002, 6–35).

On this record, Patent Owner appears to have identified adequate written description support for the limitations directed to: “multiple processing elements”; and “wherein each of the plurality of tasks is decomposable into two or more related threads” recited in proposed substitute claim 28. Petitioner does not argue otherwise. *See generally* Opp.

With respect to the limitation “wherein at least one processing element is configured to receive tasks from each engine” in proposed substitute claim 28, Patent Owner asserts the limitation is supported by at least paragraphs 27–32, 35, 43, and 50–54, and Figure 2 of the ’134 application. Mot. 8–9 (citing Ex. 1002, 6–35). Patent Owner submits that the originally filed disclosure of the ’381 patent describes partitioning the shader core’s processing elements *in time* (*temporal partitioning*) whereby

¹ The original specification of U.S. Application No. 12/874,134 is at pages 6–35 of Exhibit 1002 (which includes the prosecution history of the ’381 patent).

one way to execute tasks simultaneously in the patented invention is for *the processing elements of the shader core to receive and execute tasks from one engine at one time, and to receive and execute tasks from another engine at another time, and that over time the processing elements will receive tasks from each engine.*

Mot. 9.

Petitioner contends “at least one processing element is configured to receive tasks from each engine” in claim 28 introduces new matter and lacks written description support. Opp. 2–11. In particular, Petitioner argues

Temporal partitioning [disclosed by the ’134 application] is disclosed as one way for the shader core to “simultaneously process tasks from two or more different virtual engines.” Because temporal partitioning is an example of *simultaneous* execution of multiple tasks from *different* engines, it necessarily *excludes* . . . examples showing the entire shader core processing tasks from one engine at one point in time, and a different engine at a different point in time, which is also inconsistent with the invention’s goal of efficient multithreaded processing.

Id. at 9. We are unpersuaded by this attorney argument because it appears to be inconsistent with the disclosure of the ’134 application that temporal partitioning involves processing different tasks from different engines at different times rather than simultaneously. Ex. 1002, 7 (¶ 30), 12 (¶ 52).² Patent Owner’s argues that “the processing elements of the shader core [] receive and execute tasks from one engine at one time, and [] receive and execute tasks from another engine at another time, and[] over time the processing elements will receive tasks from each engine.” Mot. 9.

We also are unpersuaded by Petitioner’s argument that proposed substitute claim 28 lacks support in the originally filed disclosure because

[t]he “at least one processing element” of claim element [28[e]] could be in addition to “the first plurality of processing elements” and “the second plurality of processing elements” of claim elements 15[b] and [c]. Moreover, the “tasks” recited in claim

² We reference pages in Exhibit 1002 followed by the relevant paragraph(s) of the original specification found at those pages.

element [28][e] need not be part of “the plurality of tasks” of independent claim 15 and may be independent of the “first task” and the “second task” of claim 15 and claim elements [15][b] and [c].

Opp. 11. Although the recitations of “at least one processing element” and “tasks” (in “at least one processing element is configured to receive tasks from each engine” in claim 28) may present indefiniteness issues (as discussed *infra*), we do not find that Petitioner’s argument at page 11 of the Opposition identifies a written description support issue.

On this record, however, Patent Owner still does not appear to have sufficiently identified adequate written description support for proposed substitute claim 28 for the following two reasons.

First, claim 28 recites “at least *one processing element* is configured to *receive tasks from each engine*.” Mot. App. A 2 (Claim Listing) (emphasis added). Thus, claim 28 requires *one processing element (singular)* to be configured to receive a *task* from an engine, and another *task* from another engine (at the same time or at different times). The ’134 application appears to support plural *processing elements such as SIMDs (single-instruction, multiple-data devices)* receiving tasks from two or more virtual engines during temporal partitioning of the shader core’s resources. See Ex. 1002, 12 (¶ 30), 16–17 (¶¶ 50, 52). In particular, the paragraphs describing temporal partitioning in the ’134 application refer to *plural processing elements or SIMDs*, rather than describing *a single processing element or SIMD*, receiving tasks from two or more different virtual engines.

The ’134 application also describes “resources (e.g., SIMDs) of the GPU shader core can be divided between concurrent tasks based on demand, priority, and/or preset limits—while temporarily enabling any one task to (substantially) fully consume the resources of the GPU.” Ex. 1002, 12–13 (¶ 31); see also *id.* at 20–21 (¶ 64) (“scheduling module 434 may divide the resources of shader core 214 between concurrent tasks 404 based on demand, priority, and/or preset limits—while temporarily enabling any one of tasks 404 to fully consume the resources of GPU 110.”). Again, this disclosure indicates that *plural processing elements/SIMDs* can receive separate tasks such that “any one task” may “(substantially) fully consume the resources of the GPU” at a given time. See *id.* However, it is

not clear, on this record, that a single processing element “is configured to receive tasks from each engine” as recited in element 28[e].

The above-referenced paragraphs in the ’134 application reference *plural* “processing elements” or “SIMDs,” but do not describe what *a single processing element or SIMD does*; for example, the ’134 application does not appear to specify whether *one processing element* may receive *an entire task*, or whether *plural processing elements receive one task and share pieces of the task between themselves*. See generally Ex. 1002, 6–35. On the other hand, proposed substitute claim 28 requires that “at least *one processing element* is configured to *receive tasks from each engine*”—such that claim 28 requires *one processing element (singular)* to be configured to receive *an entire task* from an engine. As Petitioner notes, to support this limitation of claim 28, “the specification must provide written description support *for a single processing element*” being configured as claimed (see Opp. 6). Patent Owner has the burden to establish written description support in the ’134 application for *one processing element* being configured as recited in proposed substitute claim 28. On the current record, however, Patent Owner has provided evidence and arguments only with respect to *plural processing elements*. See Mot. 8–9. Thus, Patent Owner has not met its burden on this record.

Claim 28 also lacks written description support for “at least one processing element is configured to receive tasks from each engine” which indicates a structure of processing elements that receives tasks from each engine. The ’134 application describes an “array of processing elements is organized into single-instruction, multiple-data (SIMD) devices” and “[p]rocessing elements 220 are arranged as SIMD devices” to enable shader core 214 to process tasks from the plurality of virtual engines (Ex. 1002, 7 (¶6), 16–17 (¶ 50)), but the ’134 application describes shader core 214 and processing elements receiving tasks from arbitration logic 222 or scheduling module 434 not engines. “[A]rbitration logic 222 . . . provides tasks to shader core 214” and “[a]rbitration logic 222 includes a plurality of multiplexers to provide the tasks from the different virtual engines of command processor 230 to shader core 214.” Ex. 1002, 15–16 (¶¶ 46, 49). “[A] scheduling module 434 of GPU 110 schedules the tasks to be executed by shader core 214 based on” the priority type, relative priority between tasks processed by virtual engines 432, and availability of resources in shader core 214. *Id.* at 20–21 ¶ 64). It is not clear on this record how multiple multiplexers distribute tasks from multiple virtual engines to processing elements 220

of shader core 214 so that “at least one processing element is configured to receive tasks from each engine” as claimed on this record.

Petitioner also argues persuasively, on this record, that Figure 2 and its associated written description does not show any configuration between any processing elements and any engine.

However, Figure 2 does not show *any* configuration between any processing element (or set of processing elements) and any engine, much less *each* engine. Figure 2 only shows a two-dimensional array of a scalable number of processing elements. These disclosures indicate that the patentee had *no intended configuration at all* for how any individual processing element or groups of processing elements relate to each engine.

Opp. 7. Patent Owner has the burden to establish written description support in the '134 application for “at least one processing element *is configured to receive* tasks from each engine” as claimed in proposed substitute claim 28. On the current record, the evidence of written description support in the originally filed disclosure for *processing element(s) being configured* as claimed is ambiguous at best. Thus, Patent Owner has not met this burden.

For purposes of this preliminary guidance and based on the current record, Patent Owner does not appear to have identified an adequate written description support for proposed substitute claim 28.

We acknowledge that Patent Owner has not yet had the opportunity to respond. Patent Owner will have the opportunity to do so in its Reply to Petitioner’s Opposition (or in a revised motion to amend) in this proceeding. *See Notice, passim.*

Proposed Substitute Claim 29

On this record, Patent Owner appears to have sufficiently identified adequate written description support for proposed substitute claim 29.

Proposed substitute dependent claim 29 amends original dependent claim 17 to further recite that “the plurality of engines are arranged in parallel with one another,” “each engine is configured to receive at least one of the plurality of tasks via a distinct command buffer for each engine,” “each of the plurality of tasks is decomposable into two or more related threads,” and “each engine is configured to load state data associated with at least

one of the plurality of tasks.” Mot. App. A 2 (Claim Listing). Patent Owner indicates that these amendments are supported by paragraphs 5–8, 13, 15, 16, 27, 29, 39, 43, 53, 63, 64, Abstract, and Figures 2, 3A, 3B, and 4 of the ’134 application. Mot. 8, 10–12 (citing Ex. 1002, 6–35).

Petitioner contends the limitation “*each engine is configured to load state data associated with at least one of the plurality of tasks*. . . . introduces new matter and lacks written description support.” Opp. 12. In particular, Petitioner argues that “[t]here is no disclosure in the application for the ’381 patent of a configuration where *each* engine is *configured to* (designed to) load state data associated with at least one of the plurality of tasks. The only disclosure is that *the plurality of engines—as a whole*—loads state data.” *Id.* Petitioner asserts that the Abstract and Brief Summary both state, “The *plurality* of virtual engines is configured to . . . load a set of state data associated with each of the plurality of tasks.” *Id.* at 13 (quoting Ex. 1001, Abstr. and 3:3–6). Petitioner argues that this can be satisfied by only one of the plurality of engines loading state data. *Id.* at 13–14.

We do not find Petitioner’s contentions persuasive based on the current record because the originally-filed specification of the ’134 application and its originally-filed claims disclose “a plurality of virtual engines configured to (i) receive, from an operating system (OS), a plurality of tasks substantially in parallel with each other and (ii) load a set of state data associated with each of the plurality of tasks.” Ex. 1002, 9–10 (¶ 13), 25 (original claim 1), 26 (original claim 7), 26–27 (original claim 13), 29 (Abstr.). Each virtual engine receives a task(s) from a particular command buffer and executes those tasks on shader core 214. *Id.* at 15 (¶ 43), 20–21 (¶¶ 63, 64). Dr. Hart testifies that a skilled artisan would understand that each engine operates independently to receive tasks and load state data associated with the task it receives and transmits to the shader core. Ex. 2008 ¶ 46. On this record, we are not persuaded by Petitioner’s arguments that the description in the ’134 application indicates that only one of the plurality of engines loads state data. *See* Opp. 13–14.

Based on the current record, it appears that there is adequate support in the original disclosure of the ’381 patent for “each engine is configured to load state data associated with at least one of the plurality of tasks” (and the other limitations) recited in proposed substitute claim 29.

B. Patentability

For the reasons discussed below, at this stage of the proceeding, and based on the current record,³ it appears that Petitioner and/or the record has shown a reasonable likelihood that proposed substitute claims 28 and 29 are unpatentable.

Does the record establish a reasonable likelihood that the proposed substitute claims are unpatentable?

I. Indefiniteness

Yes, as to proposed substitute claim 28.

We have reviewed the language of proposed substitute claim 28 and “a first plurality of processing elements,” “a second plurality of processing elements,” and “at least one processing element” appear to be indefinite under 35 U.S.C. § 112, second paragraph.

Proposed substitute claim 28 recites “the shader core comprises: *multiple processing elements*,” “*a first plurality of processing elements* [are] configured to execute the first task,” “*a second plurality of processing elements* [are] configured to execute the second task,” and “*at least one processing element* is configured to receive tasks from each engine.” Mot. App. A 1–2 (Claim Listing) (emphases added). The relationship of these processing elements is unclear. It is unclear whether “a first plurality of processing elements” and “a second plurality of processing elements are elements of the “multiple processing elements” or unrelated elements. It also is unclear whether “at least one processing element is configured to receive tasks from each engine” in element 28[e] is one of the “multiple processing elements,” “a first plurality of processing elements,” “a second plurality of processing elements,” and/or some other independent element.

It also is unclear whether “the plurality of tasks” that are decomposable into one or more threads in element 28[d] and are the same or different than the “first task” and “second task” in claim 28, the tasks that “at least one processing element is configured to receive . . . from each engine” in

³ We express no view on the patentability of original claims 15–20 in this Preliminary Guidance. Instead, we focus on limitations added to the challenged claims in the Patent Owner’s Motion to Amend.

element 28[e], or “*a plurality of tasks*” received by “a plurality of engines” in claim 15.

Thus, it appears, on this record, that proposed substitute claim 28 fails to comply with the requirements of 35 U.S.C. § 112, second paragraph.

We acknowledge that Patent Owner has not yet had the opportunity to address our analysis above regarding proposed substitute claim 28 failing to comply with the requirements of 35 U.S.C. § 112, second paragraph. Patent Owner will have the opportunity to do so in its reply to Petitioner’s Opposition (or in a revised motion to amend) in this proceeding.

II. Obviousness

Yes, as to proposed substitute claims 28 and 29.

Petitioner has asserted the following obviousness challenges against the following proposed substitute claims:

- Ground asserting Jiao (Ex. 1005)⁴ against proposed substitute claim 28; and
- Ground asserting Wilt (Ex. 1004)⁵ against proposed substitute claims 28 and 29.

See Opp. 15–25.

On this record, it appears that Petitioner (or the record) has shown a reasonable likelihood that each of Jiao and Wilt renders the respective proposed substitute claims unpatentable.

Proposed Substitute Claim 28

Proposed substitute claim 28 amends original claim 16 to recite further that “each of the plurality of tasks is decomposable into two or more related threads” and “at least one processing element is configured to receive tasks from each engine.” Mot. App. A 1–2 (Claim Listing).

To provide context for the claimed “tasks” and “threads” recited in element 28[d], Patent Owner asserts that this amendment is supported by paragraphs 5–8 in the original specification of the ’381 patent (the ’134 application). *See* Mot. 8 (citing Ex. 1002 ¶¶ 5–8).

⁴ US 8,319,774 B2, filed November 29, 2011, issued November 27, 2012.

⁵ US 8,151,095 B1, filed July 18, 2008, issued April 3, 2012.

We have reviewed the relevant portions of the '134 application and note they provide that

[a] graphics-processing *task* performed by a GPU *typically involves complex mathematical computations, such as matrix and vector operations. To perform a single graphics-processing task, a GPU may execute a plurality of different threads (sequence of instructions). Each thread may comprise a shader program, such as a geometry shader, a pixel shader, a vertex shader, or the like. Each thread (e.g., shader program) is typically associated with a set of state data (such as texture handles, shader constants, transform matrices, or the like) that is locally stored in data-storage units of the GPU. The locally stored state data is called a context.*

See Ex. 1002, 7 (¶ 5) (emphases added). With respect to the term “task,” the '134 application provides examples of “*data-parallel processing tasks (e.g., graphics-processing tasks and general-compute tasks)*” and explains that “*each processing task of a given stream may include a plurality of individual processing threads.*” *Id.* at 11–12 (¶ 27), 14–15 (¶ 41) (emphases added). The '134 application describes examples in which

processing elements 220 may implement *a plurality of different shader programs (e.g., geometry shader, vertex shader, pixel shader, tessellation shader, or the like) to complete a single graphics-processing task* provided by the OS scheduler. The intermediate results of the different shader programs are sent back to vertex analyzer 208 and/or scan converter 212 and eventually recirculated to processing elements 220. After processing elements 220 have completed a task provided by the OS scheduler, the final results are provided to output logic 224.

...

streams of processing tasks are generated by a single application 302. Application 302 may be, for example, an end-user application that generates graphics-processing tasks (such as a video-game application, a computer-aided design (CAD) application, or the like) or an end-user application that generates general-compute tasks (e.g., mathematical algorithms, physics simulations, or the like) to be executed on a GPU.

Id. at 17 (¶ 54), 18–19 (¶ 58) (emphases added).

Petitioner's obviousness grounds assert each of Jiao and Wilt against claim 28. Thus, Petitioner relies on each of Jiao and Wilt to teach or suggest that "each of the plurality of tasks is decomposable into two or more related threads" and "at least one processing element [that] is configured to receive tasks from each engine," as recited in proposed substitute claim 28. We analyze each of Petitioner's grounds below.

Ground relying on Jiao

amendment directed to "each of the plurality of tasks is decomposable into two or more related threads" in claim 28

Petitioner contends that Jiao's *execution units (EUs) 420a through 420h* teach "processing elements" (of a "shader core"), as recited in proposed substitute claim 28. Opp. 15, 17–18 (citing Ex. 1005, 1:43–47, 8:15–54, 9:33–35, Fig. 4). Petitioner argues that Jiao teaches or renders obvious the claimed shader core and its multiple processing elements receiving tasks from engines where "each of the plurality of tasks is decomposable into two or more related threads," because the "'engines' of Jiao (e.g., the vertex shader, geometry shader, and pixel shader) deliver tasks to the execution units 420" each of which "is capable of multi-threaded operations simultaneously," and "the EU pool control unit 206 handles the assigning of those tasks to available threads within the various execution units 420." Opp. 16–17 (citing Ex. 1005, 1:43–47, 8:15–54, 9:25–54, Fig. 4). Petitioner contends that "[b]ecause [Jiao's] 'tasks' can encompass a very wide range of programmable operations to meet the needs of different computer programs (e.g., [Jiao], 5:47–51), it is inherent, or at least obvious to a POSITA, that each of a plurality of 'tasks' is decomposable into two or more related threads." Opp. 17 (citing Paper 1, 43; Ex. 1003 ¶ 10).

Jiao appears to support Petitioner's contentions. For example, Jiao discloses that "shader tasks [are] associated with various shader programs, including a vertex shader, geometry shader, and/or pixel shader, processing data for the graphics pipeline 208." Ex. 1005, 5:47–51. Jiao explains that a programmable vertex processor of vertex shader 304 performs *operations* of "vector space transformations and . . . other computations," such "vertex transformations" and "operations such as transformations, skinning, and lighting"; geometry shader 306 performs *operations* such as providing "multiple vertices that form a single topology" and "perform[ing] various algorithms, such as tessellation, [and] shadow volume generation"; and pixel shader 310 performs *operations*

such as “interpolations and other operations that collectively determine pixel colors.” *See id.* at 1:59–62, 4:2–5, 5:47–51, 6:21–25, 6:29–35, 6:40–44, 7:56–67.

Jiao’s description of the *operations* implemented by vertex shader 304, geometry shader 306, and pixel shader 310 appears to be *commensurate with the description of exemplary “tasks” in the ‘381 patent*—such as a “graphics-processing task . . . [that] typically involves complex mathematical computations, such as matrix and vector operations” or “general-compute tasks (e.g., mathematical algorithms[])” (*see* Ex. 1001, 1:46–49, 9:24–25).

Jiao further discloses that the *operations/tasks* implemented by vertex shader 304, geometry shader 306, and pixel shader 310 are performed in accordance with various shader *programs*. *See* Ex. 1005, 1:47–50, 4:3–9, 5:47–51, 6:56–64. For example, Jiao provides that

[t]he computational core 204 comprises a pool of multiple execution units to meet the computing requirements imposed by *shader tasks associated with various shader programs, including a vertex shader, geometry shader, and/or pixel shader, processing data for the graphics pipeline 208.*

...

the computational core 204 . . . processes various instructions and includes the CB [constant buffer] system 200. Various types of *shader programs can be executed or mapped to the computational core 204, including the vertex shader 304, geometry shader 306, and pixel shader 310.* That computational core 204, for multi-issue processors, is capable of *processing multiple instructions within a single clock cycle.*

Id. at 5:47–51, 6:56–64 (emphases added). As Petitioner asserts, Jiao also describes *threads* and provides that “*each thread fetches an instruction according to a program counter.*” *Id.* at 9:12–14 (emphasis added). Jiao’s description of EU pool control unit 206 assigning shader tasks to available *threads* within execution units 420 with “*each thread fetch[ing] an instruction according to a program counter,*” appears to be commensurate with the description, in the ‘381 patent, of *threads* as a “*sequence of instructions*” that are executed in order to perform a task consisting of multiple complex operations. *See* Ex. 1001, 1:46–52 (emphasis added); Ex. 1005, 9:12–14 (emphasis added).

Further, with respect to the execution units (EUs) 420, Jiao discloses that shaders 304, 306, and 310 perform their received tasks by generating subtasks for EUs 420 to perform. Ex. 1005, 5:17–57, 7:56–8:34, 9:7–54. In particular, Jiao provides that “[a]s *various shader tasks* are generated by the vertex shader 304, geometry shader 306, and pixel shader 310, they are *delivered to the respective execution units 420* to be carried out,” whereby “EU pool control unit 206 functions as a global scheduler *for the tasks* and *assigns appropriate threads in the execution units 420* using a data-driven approach (e.g., vertex, pixel, geometry packets in the input)” such that “[e]ach execution unit **420** of the EU pool **414** may *store a plurality of tasks or threads (e.g., in some embodiments, thirty-two threads) at the same time.*” *Id.* at 9:10–20, 9:35–42 (emphases added). Thus, “[a]s *individual tasks* are generated, the EU pool control unit **206** *handles the assigning of those tasks to available threads* within the various execution units **420**” and “[a]s *tasks* are completed, the EU pool control unit **206** further manages the release of the *relevant threads.*” *Id.* at 9:39–46 (emphases added).

Because Jiao discloses (i) *operations/tasks* being implemented by vertex/geometry/pixel shaders in accordance with *shader programs*, and (ii) “assigning vertex shader, geometry shader, and pixel *shader tasks* to *threads* of the various execution units” where “*each thread fetches an instruction according to a program counter,*” we agree with Petitioner’s assessment that “it is . . . at least obvious to a POSITA, that each of a plurality of ‘tasks’ [in Jiao] is decomposable into two or more related threads.” Opp. 17; *see* Ex. 1005, 1:47–50, 4:3–9, 5:47–51, 6:56–64, 9:12–14, 9:39–46 (emphases added). The ’381 patent describes a thread as a sequence of instructions of a program such as a shader program. Ex. 1001, 1:49–53. Jiao similarly describes execution units executing threads of tasks of shader programs by fetching instructions to execute.

In the Motion to Amend, Patent Owner presents arguments directed to “each of the plurality of tasks is decomposable into two or more related threads” recited in proposed substitute claim 28. Mot. 17–18. Patent Owner argues that “there is no disclosure [in Jiao] of a task that is decomposable into two or more related threads” because Jiao’s EU pool control unit 206 merely “assign[s] *threads* to open *thread slots*” and the “so-called ‘tasks’ [in Jiao] *are what the ’381 Patent would refer to as a thread.*” *Id.* (citing Ex. 2008 ¶¶ 64–65) (emphases added). In particular, Patent Owner submits that

although Jiao refers to the shaders generating “tasks” that are then received by the execution units in the alleged shader core, the other descriptions make clear that these so-called “tasks” are what the ’381 Patent would refer to as a thread. For example, although Jiao states that “[a]s various shader tasks are generated by the vertex shader 304, geometry shader 306, and pixel shader 310, they are delivered to the respective execution units 420 to be carried out,” the further detail regarding how that happens clarifies that those “tasks” are in fact threads. Ex. 1005 at 9:35–38. Jiao states in the next sentence that “the EU pool control unit 206 handles the assigning of those tasks *to available threads* within the various execution units 420.” *Id.* at 9:39–41. Moreover, Jiao states “EU pool control unit 206 functions as a global scheduler for the tasks and assigns appropriate threads in the execution units 420 using a data-driven approach (e.g., vertex, pixel, geometry packets in the input). For instance, the EU pool control unit 206 assigns a thread to one of the empty thread slots in the respective execution unit 420 of the EU pool 414.” *Id.* at 9:15–21.

Mot. 17–18 (citing Ex. 2008 ¶¶ 64–65).

At this stage and based on the current record, we are unpersuaded by Patent Owner’s arguments. We are unpersuaded as we do not see Jiao as *identifying its shader “tasks”* (tasks generated by vertex, geometry, and pixel shader programs) as *“threads”* (as Patent Owner argues, *see supra*).

More particularly, Jiao discloses *tasks* and *threads* and distinguishes them as (i) *tasks* are associated with or generated by vertex/geometry/pixel shader programs, the *tasks* belong to a rendering context (which defines visual properties of a graphical object to be rendered), and the EUs are designed to “meet the computing requirements imposed by [the] shader tasks” (*see* Ex. 1005, 3:60–67, 5:47–51, 6:49–50, 9:5–7, 9:35–38), while (ii) *threads* are executed by “the various execution units [(EUs)] 420” with “each thread fetch[ing] an instruction according to a program counter,” some *threads* are “available threads within the various execution units” while other “threads . . . are occupied,” and the EU pool control unit maintains “a resource table of threads and memories for all execution units” and “updates its resource table to mark [a] thread as free and to add the amount of total thread common register file space back to the available space” (*see id.* at 9:12–15, 9:39–54, 9:67–10:3).

When Jiao mentions task(s) and thread(s) in the same sentence, Jiao provides that: “the EU pool 414 may *store a plurality of tasks or threads* (e.g., in some embodiments, *thirty-two threads*) at the same time”; “EU pool control unit 206 functions as a global *scheduler for the tasks* and *assigns appropriate threads in the execution units 420*” such that “[a]s individual tasks are generated, the EU pool control unit 206 handles *the assigning of those tasks to available threads within the various execution units 420*”; “when a *task is assigned* to one execution unit . . . EU pool control unit 206 will *mark the thread as busy* and subtract the total available common register file memory by the amount of the appropriate register file *footprint for each thread*” and “[a]s *tasks are completed*, the EU pool control unit 206 further manages the *release of the relevant threads*”; EU pool control unit 206 also performs “‘bookkeeping’ *of the tasks and threads*” and “*knows which threads have been assigned tasks and are occupied*”; and “[w]hen a *thread completes its assigned task(s)*, the execution unit 420 *running the thread* sends an appropriate signal to the EU pool control unit” which “*updates its resource table to mark the thread as free* and to add the amount of total thread common register file space back to the available space.” *See id.* at 9:10–17, 9:39–41, 9:39–61, 9:65–10:3 (emphases added).

Jiao also discloses that “each thread fetches an instruction according to a program counter.” Ex. 1005, 9:13–14. The ’381 patent also describes threads as a sequence of instructions used to perform a task generated by a particular shader program such as a geometry shader, a pixel shader, and a vertex shader. Ex. 1001, 1:49–53.

As shader programs generate different tasks, Jiao’s EU pool control unit 206 *decomposes/assigns/delivers* the different tasks from different shader programs to available thread slots in respective execution units to be carried out, and as tasks are completed by various execution units 420, the threads of those execution units 420 are released to perform other tasks generated by various shader programs. *Id.* at 9:15–47; *see also id.* at 7:65–67 (describing “known *tasks* such as bounding box generation, culling, edge function generation, and triangle level rejections, among other known functions” (emphasis added)), 9:15–19 (“EU pool control unit 206 functions as a global scheduler for the tasks and assigns appropriate threads [for each task] in the execution units 420 using a data-driven approach (e.g., vertex, pixel, geometry packets in the input).”).

In the Motion to Amend, Patent Owner additionally argues that “there is no disclosure of simultaneous execution of different tasks [in Jiao] because *there is nothing to indicate any two threads correspond to different parent tasks or that each of the alleged tasks is decomposable into more than one thread.*” Mot. 18 (citing Ex. 2008 ¶ 66) (emphasis added). As discussed *supra*, on this record, we are persuaded that Jiao teaches or suggests tasks that are “decomposable into two or more related threads” as recited in proposed substitute claim 28). Jiao discloses that, at any given time, there may be multiple rendering contexts (different sets of state data) allowed in each execution unit 420, and this rendering context information is passed from EU pool control unit 206 before tasks belonging to this context are started in an execution unit 420. Ex. 1005, 8:67–9:10. Then, each execution unit 420 of EU pool 414 may store a plurality of tasks or threads at the same time for such rendering context information and each thread fetches an instruction to perform to complete a particular task associated with that rendering context information. *Id.* at 9:11–24. EU pool control unit 206 assigns an appropriate number of threads in execution units 420 for each task and as various shader tasks are generated, they are delivered to respective execution units 420, which are capable of multi-threaded operations simultaneously to carry out an assigned task. *Id.* at 9:33–38.

Thus, at this stage of the proceeding and on this record, it appears that Petitioner has sufficiently shown that Jiao teaches or suggests “each of the plurality of tasks is decomposable into two or more related threads,” as recited in proposed substitute claim 28.

We acknowledge that Patent Owner has not yet had the opportunity to address Petitioner’s discussion of Jiao, and our analysis above, with respect to the claimed “threads” and “tasks.” Patent Owner will have the opportunity to do so in a Reply to Petitioner’s Opposition (or in a Revised Motion to Amend) in this proceeding. *See* Notice, *passim*.

We also acknowledge Patent Owner’s additional argument that “the alleged shaders [of Jiao] do not receive instructions or threads, much less tasks; they only receive data, in the form of primitives and constants.” *See* Mot. 17. This argument appears to pertain to the “engines” features in originally challenged claims 15 and 17 (e.g., “a plurality of engines configured to receive, from a scheduling module associated with a second processing unit, a plurality of tasks” in original claim 15), and to the original grounds asserted against those original claims. The Preliminary Guidance, however, focuses on the proposed substitute claims, and

specifically on the amendments proposed in the Motion. *See* Notice, 84 Fed. Reg. at 9,497 (“The Board’s preliminary guidance will focus on the limitations added in the patent owner’s MTA, and will not address the patentability of the originally challenged claims.”), 9,500 (“The Board’s preliminary guidance will focus on the limitations added in the Patent Owner’s motion to amend, and will not address the patentability of the originally challenged claims.”) As noted above, we do not address here the patentability of (and the original grounds asserted against) originally challenged claims. *Id.*

amendment directed to “at least one processing element is configured to receive tasks from each engine” in claim 28

Petitioner also contends that Jiao teaches or suggests “at least one processing element [being] configured to receive tasks from each engine” because “Jiao discloses or at least renders obvious that at least one EU [in Jiao] (a processing element) is configured to receive tasks from each engine [(of the engines including vertex shader 304, geometry shader 306, and pixel shader 310 of Jiao)].” Opp. 17–18. Petitioner asserts that is so because

Jiao discloses that as “various shader tasks are generated by the vertex shader 304, geometry shader 306, and pixel shader 310, they are delivered to the respective execution units 420 to be carried out.” Ex. 1005, 9:35–39. Specifically, “the EU pool control unit 206 is responsible for assigning vertex shader, geometry shader, and pixel shader tasks to threads of the various execution units 420,” depending on “how much free space is available for each execution unit.” *Id.*, 9:39–10:9. Since the EU pool control unit assigns tasks from *each* of the plurality of engines to *each* of the Execution Units (e.g., eight EUs), and each Execution Unit is a “processing element,” it is obvious that at least one processing element (EU) is configured to receive tasks from each engine. This is also shown in Figure 4 [of Jiao], in which each processing element (EU0 to EU7, comprising 8 processing elements) is shown as receiving tasks from each of the three engines VS 304, GS 306, and PS 310.

Opp. 18.

Jiao appears to support Petitioner’s contentions. For example, Jiao discloses that *EUs 420 are similarly configured*, as “[e]ach of the execution units 420 comprises an embodiment of the CB [(constant buffer)] system 200,” “[e]ach of these execution units 420 is identical in design and configurable for programmed operation,” and “each execution unit 420 is capable of multi-threaded operations simultaneously.” See Ex. 1005, 8:26–29, 9:31–35. The assignment of work/tasks to EUs 420 is done based on “a data-driven approach (e.g., vertex, pixel, geometry packets in the input)” —which, Jiao explains, relies on the availability of the EUs’ memory and thread slots as recorded in a “resource table of threads and memories for all execution units 420” maintained by the EU pool control unit. *Id.* at 9:15–18, 9:18–21 (“EU pool control unit 206 assigns a thread to one of the empty thread slots in the respective execution unit 420 of the EU pool 414.”), 9:46–49. Jiao also provides that “[a]s various *shader tasks are generated by the vertex shader 304, geometry shader 306, and pixel shader 310, they are delivered to the respective execution units 420* to be carried out.” *Id.* at 9:35–38 (emphasis added); see also *id.* at 5:46–51 (“computational core 204 comprises a pool of *multiple execution units to meet the computing requirements imposed by shader tasks associated with various shader programs, including a vertex shader, geometry shader, and/or pixel shader, processing data for the graphics pipeline 208*” (emphasis added)), 9:43–47 (“EU pool control unit 206 is responsible for assigning vertex shader, geometry shader, and pixel shader tasks to threads of the various execution units 420”). Thus, “[a]s individual tasks are generated, the EU pool control unit 206 handles the assigning of those tasks to available threads within the various execution units 420” based on the EU pool control unit’s knowledge of “which threads have been assigned tasks and are occupied, which threads have been released after thread termination, how many common register file memory registers are occupied, and how much free space is available for each execution unit.” *Id.* at 9:39–54.

Thus, Jiao describes the EUs (units 420) receiving tasks from the multiple engines (vertex shader 304, geometry shader 306, and pixel shader 310) based on each EU’s available processing power/memory as Petitioner asserts. Opp. 18. We also agree with Petitioner’s assessment that “Jiao discloses or at least renders obvious that at least one EU (a processing element) is configured to receive tasks from each engine” among vertex shader 304, geometry shader 306, and pixel shader 310. *Id.*

Thus, at this stage of the proceeding, it appears that Petitioner has sufficiently shown that Jiao teaches or suggests “at least one processing element [being] configured to receive tasks from each engine,” as recited in proposed substitute claim 28.

We acknowledge that Patent Owner has not yet had the opportunity to address Petitioner’s discussion of Jiao (and our analysis above) with respect to the claimed “at least one processing element [being] configured to receive tasks from each engine.” Patent Owner will have the opportunity to do so in a Reply to Petitioner’s Opposition (or in a Revised Motion to Amend) in this proceeding. *See Notice, passim.*

In conclusion, based on the current record, at this stage of the proceeding, it appears that Petitioner has shown a reasonable likelihood that proposed substitute claim 28 is rendered obvious by Jiao.

Ground relying on Wilt

amendment directed to “each of the plurality of tasks is decomposable into two or more related threads” in claim 28

Petitioner contends that Wilt’s *processing cores 208* teach the claimed “processing elements” (of a “shader core”), as recited in proposed substitute claim 28. Opp. 19 (citing Ex. 1004, 4:1, Fig. 2). Petitioner also argues that Wilt teaches or renders obvious the claimed shader core and its multiple processing elements receiving tasks from engines where “each of the plurality of tasks is decomposable into two or more related threads,” because Wilt’s multithreaded processing cores 208 are assigned tasks “relating to a wide variety of applications,” and the cores process the tasks by “executing a large number (e.g., tens or hundreds) of threads concurrently.” Opp. 19–21 (citing Ex. 1004, 3:60–4:25, 4:1–19, 4:33–40, 4:61–5:4, 5:59–6:6, 6:13–27, 8:15–29, 10:21–51). Petitioner contends

[Wilt’s] “tasks” encompass a very wide range of programmable operations. It is therefore inherent, or at least obvious to a POSITA, that when a processing “task” [in Wilt] is assigned to a multithreaded processing “core” with tens or hundreds of threads for execution, that each of a plurality of “tasks” is decomposable into two or more related threads.

Opp. 20–21.

Wilt appears to support Petitioner’s contentions. For example, Wilt discloses that “[w]hen a *new processing task* arrives, work distribution unit 210 *assigns the task to a core 208* that is asserting the ready signal.” Ex. 1004, 4:15–18 (emphases added). The processing *tasks* that cores 208 can be programmed to execute are

tasks relating to a wide variety of applications, including but not limited to linear and nonlinear data transforms, filtering of video and/or audio data, modeling operations (e.g., applying laws of physics to determine position, velocity and other attributes of objects), image rendering operations (e.g., vertex shader, geometry shader, and/or pixel shader programs). . . .
. . . [and] various tasks related to generating pixel data from graphics data supplied by CPU 102.

Id. at 4:33–52 (emphases added). “Each processing core 208 is capable of executing a large number (e.g., tens or hundreds) of threads concurrently, where each thread is an instance of a program” (more particularly, “an instance of a particular program executing on a particular set of input data”). *Id.* at 4:1–7, 5:63–66. After a core 208 is assigned a task, the core may execute the task by

execut[ing] a series of common instructions on the parallel processing engines 302 within the core 208. *The series of instructions to a single parallel processing engine 302 constitutes a thread, as defined previously, and the collection of a certain number of concurrently executing threads among the parallel processing engines 302 within a core 208 is referred to herein as a “warp” or “thread group.”* Additionally, a plurality of *related thread groups* may be active (in different phases of execution) at the same time on a core 208. *This collection of thread groups is referred to herein as a “cooperative thread array” (“CTA”).*

Id. at 6:14–27 (emphases added). Wilt provides examples of thread groups (such as SIMT thread groups and CTAs) that execute in a core 208. With regard to *SIMT (single-instruction multiple-thread) thread groups*, Wilt provides that

[b]ecause instruction unit 312 may issue the same instruction to all P processing engines 302 in parallel, core 208 is advantageously used to process threads in “SIMT thread groups.” As used herein, a “SIMT thread group” or “warp” refers

to a group of up to P threads of execution of the same program on different input data, with one thread of the group being assigned to each processing engine 302. . . .

. . . Thus, in a given cycle, all processing engines 302 in core 208 are nominally executing the same instruction for different threads in the same SIMT thread group. *In some instances, some threads in a SIMT thread group may be temporarily idle, e.g., due to conditional or predicated instructions, divergence at branches in the program, or the like.*

Id. at 7:43–57, 7:65–8:5 (emphasis added). And with regard to CTAs, Wilt provides that

[i]n some embodiments, multithreaded processing core 208 of FIG. 3 can execute general-purpose computations using thread blocks or thread arrays. . . .

. . . the thread arrays are arranged as “cooperative thread arrays,” or CTAs. *Each CTA is a group of threads that concurrently execute the same program (referred to herein as a “CTA program”) on an input data set to produce an output data set. In a CTA, the threads can cooperate by sharing data with each other in a manner that depends on thread ID. For instance, in a CTA, data can be produced by one thread and consumed by another.*

. . .

For example, as is known in the art, an array of data values (e.g., pixels) can be filtered using a 2-D kernel-based filter algorithm, in which the filtered value of each pixel is determined based on the pixel and its neighbors. In some instances the filter is separable and can be implemented by computing a first pass along the rows of the array to produce an intermediate array, then computing a second pass along the columns of the intermediate array. *In one CTA implementation of a separable 2-D filter, the threads of the CTA load the input data set (or a portion thereof) into shared memory 306, then synchronize. Each thread performs the row-filter for one point of the data set and writes the intermediate result to shared memory 306. After all threads have written their row-filter results to shared memory 306 and have synchronized at that point, each thread performs the column filter for one point of the data set. In the course of performing the column filter, each thread reads the appropriate*

row-filter results from shared memory 306, and a thread may read row-filter results that were written by any thread of the CTA. . . . *all threads in a CTA execute the same program in the same core 208.*

Id. at 8:56–60, 9:1–10, 9:63–10:15, 10:38–41 (emphases added).

Thus, Wilt provides that its multiple *processing cores 208* (“processing elements”) individually receive and execute *tasks for an application* (such as tasks that perform image rendering operations and filtering of image data), *each task comprising* its own thread group (such as a SIMT thread group or a cooperative thread array (CTA)) of *multiple, related threads* (see discussion of Wilt, *supra*).

Patent Owner, however, contends that Wilt does not disclose that the alleged shader core or processing elements execute tasks with “each of the plurality of tasks [being] decomposable into two or more related threads.” Mot. 13–16. Patent Owner argues “Wilt operates at the instruction level, or at best the thread level, not the task level” as Wilt’s “CPU only issues ‘commands’ and a ‘command stream’” and “each command stream is only one thread, not two or more [sic].” *Id.* at 14–15. Patent Owner addresses Wilt’s reference to “tasks” as follows:

Although Wilt states in some portions that cores 208 execute “tasks,” given that the only thing the alleged engines upstream of the shader core ever receive are at best threads, and there is no disclosure of creation of tasks from those threads, the only reasonable interpretation is that the cores actually execute what the ’381 Patent would call threads, not tasks as the ’381 Patent and the claim language in claim 28 describe that term. Ex. 1004 at 4:1–45.

Id. at 16. Patent Owner also contends “[t]here is no evidence that threads from two different tasks would ever be executed at the same time [in Wilt]” and “there is no evidence that what the PPU or any subcomponent [in Wilt] executes comprises more than one thread.” *Id.* at 15. Patent Owner’s support for these contentions relies upon Patent Owner’s additional arguments that

looking just at a thread or multiple threads, it is not knowable without more information (that is lacking in Wilt, Jiao, and Stuttard) which task or tasks they were decomposed from; threads could have been decomposed from, and thus correspond

to, the same task, for example. Ex. 2008 at ¶ 58; Ex. 2007 at 21:6–27:4. It is insufficient to invalidate the claims to argue that something might, could, or should happen.

Id. at 15–16.

At this stage and based on the current record, we are unpersuaded by Patent Owner’s arguments. On this record, it appears that a command stream writes commands (i.e., tasks) for a cooperative thread array (CTA) to process. Each command (task) is associated with state parameters such as the number of threads in the CTA, the location of input data set to be processed by the CTA, and the CTA program to execute to perform the task. Core interface 303 loads the state parameters into core 208 and launches threads until the number of threads needed to perform the task has been launched. Ex. 1004, 10:21–36.

We are unpersuaded that Wilt operates only at “the thread level” but “not [at] the task level,” or that “there is no evidence that what the PPU or any subcomponent [in Wilt] executes comprises more than one thread” (as Patent Owner argues, *see* Mot. 14–15). As discussed *supra* and as noted by Petitioner (Opp. 20–21), Wilt’s *processing cores 208* (“processing elements”) individually receive and execute *tasks for an application* (such as tasks that perform image rendering operations and filtering of image data), *each task comprising its own thread group (such as a SIMT thread group or a cooperative thread array) of multiple, related threads*. For example, tasks are decomposed into SIMT thread groups or warps that are groups of threads that execute different tasks of the same program by processing different input data on each thread. Ex. 1004, 7:43–49. The same instruction (i.e., task) is assigned to multiple engines 302 in parallel and core 208 processes the threads in SIMT thread groups that execute the program on different input data to execute threads of the task. *Id.* A thread array executes the same program and each thread has an identifier (ID) that determines what portion of the input data for a task each thread is to process and which portion of output data for a task each thread produces or writes. *Id.* at 8:58–67. In addition, where PPU 202 includes multiple cores 208, each core and its associated engines 302 receive and process a particular, different task. *Id.* at 8:37–44.

We therefore are unpersuaded by Patent Owner’s arguments that Wilt only discloses *unrelated and disparate* threads that *are not part of a task*. *See* Mot. 14–16 (arguing that “there is no disclosure [in Wilt] of creation of tasks from those threads [received by the engines]”). As discussed *supra*,

a *SIMT thread group* of threads executed by one core 208 in Wilt includes *related threads* whose execution is subject to “conditional or predicated instructions, divergence at branches in the program.” See Ex. 1004, 7:65–8:5 (“some threads in a SIMT thread group [executed by processing engines 302 in core 208, in a given clock cycle] may be temporarily idle, e.g., due to conditional or predicated instructions, divergence at branches in the program”). And a *cooperative thread array (CTA)* of threads executed by one core 208 in Wilt includes *related threads* that cooperate by sharing data and intermediate results. See *id.* at 9:1–9, 9:39–55, 9:63–66 (CTAs “perform computations that lend themselves to a data-parallel decomposition” with “transforms in any number of dimensions (e.g., Fast Fourier Transforms), and various filtering algorithms including convolution filters . . . [and] separable filters” such as a separable 2-D filter based on a 2-D kernel-based filter algorithm, where “threads [in a CTA] can cooperate by sharing data with each other in a manner that depends on thread ID. For instance, in a CTA, data can be produced by one thread and consumed by another.”). Moreover, Patent Owner’s argument—that “the only thing the alleged engines upstream of the shader core [in Wilt] ever receive are at best threads” and not tasks (see Mot. 16)—appears to be at variance with the following description in Wilt:

Cores 208 receive processing tasks to be executed via a work distribution unit 210, which receives commands defining processing tasks from a front end unit 212. . . . When a new processing task arrives, work distribution unit 210 assigns the task to a core 208 that is asserting the ready signal; if no core 208 is asserting the ready signal, work distribution unit 210 holds the new processing task until a ready signal is asserted by a core 208. Those skilled in the art will recognize that other algorithms may also be used and that the particular manner in which work distribution unit 210 distributes incoming processing tasks is not critical to the present invention.

Ex. 1004, 4:4–23 (emphases added).

We also are unpersuaded by Patent Owner’s arguments that “in Wilt, shader core receives and allegedly simultaneously executes at best different instructions or threads, not tasks,” and “[t]here is no evidence [in Wilt] that threads from two different tasks would ever be executed at the same time.” Mot. 14–15. We are unpersuaded by these arguments because Wilt provides that multithreaded processing cores 208

(“processing elements” of a “shader core”) each receive and execute tasks—which may include related threads in a SIMT thread group or CTA, as discussed *supra*—in order to produce “a highly parallel processor”:

Each PPU 202 advantageously implements *a highly parallel processor*. As shown in detail for PPU 202(0), a PPU 202 includes a number C of cores 208, where $C \geq 1$. *Each processing core 208 is capable of executing a large number (e.g., tens or hundreds) of threads concurrently*, where each thread is an instance of a program; one embodiment of *a multithreaded processing core 208* is described below. *Cores 208 receive processing tasks to be executed. . . . [W]ork distribution unit 210 receives a “ready” signal from each core 208 indicating whether that core has sufficient resources to accept a new processing task*. When a new processing task arrives, work distribution unit 210 *assigns the task to a core 208 that is asserting the ready signal*.

Ex. 1004, 4:1–20 (emphases added).

At this stage of the proceeding, it appears that Petitioner has sufficiently shown that Wilt teaches or suggests “each of the plurality of tasks is decomposable into two or more related threads,” as recited in proposed substitute claim 28.

We acknowledge that Patent Owner has not yet had the opportunity to address Petitioner’s discussion of Wilt (and our analysis above) with respect to the claimed “tasks . . . decomposable into two or more related threads” limitation. Patent Owner will have the opportunity to do so in a Reply to Petitioner’s Opposition (or in a Revised Motion to Amend) in this proceeding. *See Notice, passim*.

We acknowledge Patent Owner’s additional arguments that “[t]here is no disclosure of modification of these command streams [issued by Wilt’s CPU] into tasks before transmitting them to the alleged shader core, and thus, at best, this also only discloses that a shader core can receive and execute (allegedly simultaneously) multiple command streams,” and that “executing multiple command streams [in Wilt] does not indicate executing multiple tasks simultaneously.” *See Mot. 15*. These arguments appear to pertain to features in originally challenged claims 15 and 16, and to the original grounds asserted against these original claims. The Preliminary Guidance, however, focuses on the proposed substitute

claims, and specifically on the amendments proposed in the Motion. *See* Notice, 84 Fed. Reg. at 9,497 (“The Board’s preliminary guidance will focus on the limitations added in the patent owner’s MTA, and will not address the patentability of the originally challenged claims.”), 9,500 (“The Board’s preliminary guidance will focus on the limitations added in the Patent Owner’s motion to amend, and will not address the patentability of the originally challenged claims.”) As noted above, we do not address here the patentability of (and the original grounds asserted against) originally challenged claims. *Id.*

amendment directed to “at least one processing element is configured to receive tasks from each engine” in claim 28

Petitioner contends that Wilt teaches or suggests “at least one processing element is configured to receive tasks from each engine” because

Wilt’s front end 212, work distribution unit 210, and core interface 303 within each PPU collectively serve as a single “engine,” and the set of cores 208 within and across all the PPUs (PP1 ... PPU U-1) collectively serve as the “shader core.” *See* Ex. 1004, Fig. 2–3. Wilt discloses that PPUs can communicate with each other through the bidirectional communication path 113 and the memory bridge 105. *See* Ex. 1004, 4:7–23 and Fig. 3. . . .

Because bidirectional communication path 113 and memory bridge 105 connect each engine within each PPU, a POSITA would understand from Wilt’s explicit teaching that when no cores in a PPU have available resources, that its work distribution unit 210 may send a processing task back through the said communication path and memory bridge so that it can run on a different PPU which has cores signaling “ready.” *Id.*; Ex. 2007, 42:21-45:6. In this case, each of the cores 208 of a PPU is configured to receive tasks not only from the single engine associated with that PPU but also from each engine of every other PPU. *Id.*; Ex. 2007, 42:21–45:6. The bidirectional distribution of tasks is further suggested by Wilt’s teaching that “each PPU 202 includes a host interface that communicates with the rest of system 100 via communication path 113.” . . .

Opp. 21–22.

We have reviewed Petitioner's cited portions of Wilt and find that they do not support Petitioner's assertion that Wilt's communication path 113, memory bridge 105, and host interfaces 206 between PPUs and the CPU/system memory suggests "that when no cores in a PPU have available resources, that [the PPU's] work distribution unit 210 may send a processing task back through the said communication path and memory bridge so that it can run on a different PPU" or that "each of the cores 208 of a PPU is configured to receive tasks not only from the single engine associated with that PPU but also from each engine of every other PPU." *See* Opp. 21–22. We do not understand Wilt's disclosure as teaching or suggesting Petitioner's alleged transfer of tasks from one PPU's work distribution unit to another PPU's cores.

Although we disagree with Petitioner's contentions at pages 21–22 of the Opposition, we also are unpersuaded by Patent Owner's argument regarding this claim limitation at pages 16–17 of the Motion. In particular, we note that, under Petitioner's alternative contentions that Wilt discloses the claimed "plurality of engines" as *work distribution unit 210, front end unit 212, and core interface unit 303 in each PPU* (*see* Pet. 26, Inst. Dec. 16, 18), it appears that one processing element (core 208 in the PPU) is configured to receive tasks from each of engines 210, 212, and 303 (in that PPU). *See* Wilt Figs. 2–3. In particular, Wilt teaches that a core 208 *receives tasks from the serial arrangement of the front end unit 212, work distribution unit 210, and core interface 303*:

commands related to processing tasks may be directed to a front end unit 212. . . .

. . . Cores 208 receive processing tasks to be executed via a work distribution unit 210, which receives commands defining processing tasks from a front end unit 212. . . . When a new processing task arrives, work distribution unit 210 assigns the task to a core 208 that is asserting the ready signal.

. . .

Operation of core 208 is advantageously controlled via a core interface 303. In some embodiments, core interface 303 receives data to be processed (e.g., primitive data, vertex data, and/or pixel data) as well as state parameters and commands defining how the data is to be processed (e.g., what program is to be executed) from work distribution unit 210. . . . Core interface 303 also initializes each new thread or SIMT thread group in

instruction unit 312, then signals instruction unit 312 to begin executing the threads. . . . Core interface 303 can then initiate other processes, e.g., to retrieve output data from shared memory 306 and/or to prepare core 208 for execution of additional threads or SIMT thread groups.

Ex. 1004, 3:60–61, 4:7–18, 8:14–30. In the Motion to Amend, Patent Owner argues that Wilt’s “cores 208 only receive alleged tasks from core interface 303 according to Petitioner’s expert and no processing element in a PPU receives tasks from each of the alleged engines.” Mot. 16–17. Patent Owner’s argument is premised on an interpretation of the term “receive” in proposed substitute claim 28 as requiring a path of *direct* receipt—i.e., that the at least one processing element should be configured to receive tasks *directly* from each engine. At this stage and based on the current record, we are not persuaded that claim 28’s limitation of “at least one processing element is configured *to receive tasks from each engine*” precludes *indirect* receipt of the tasks from each engine. Rather, it appears that the ’381 patent contemplates processing elements that receive tasks from engines directly, or *indirectly*—through arbitration logic 222 (and also, possibly, through MUX 206, vertex analyzer 208, and scan converter 212). *See* Ex. 1001, 7:30–62, Fig. 2.

Because a processing element (core 208) in Wilt is configured to *receive tasks directly* from core interface unit 303 (associated with that core’s PPU) and *indirectly* from each of work distribution unit 210 and front end unit 212 (associated with that core’s PPU), it appears that Wilt teaches that a processing element (core 208) “is configured to receive tasks from each engine” (as recited in claim 28) among engines 210, 212, and 303 that are in the same PPU as core 208. *See* Ex. 1004, 3:60–61, 4:7–18, 8:14–30, Figs. 2–3.

We acknowledge that Patent Owner has not yet had the opportunity to address our analysis of Wilt, above, with respect to the claimed “at least one processing element [being] configured to receive tasks from each engine.” Patent Owner will have the opportunity to do so in a Reply to Petitioner’s Opposition (or in a Revised Motion to Amend) in this proceeding. *See* Notice, *passim*.

In conclusion, based on the current record, at this stage of the proceeding, it appears that Petitioner together with the current record has shown a

reasonable likelihood that proposed substitute claim 28 is rendered obvious by Wilt.

Proposed Substitute Claim 29

Proposed dependent substitute claim 29 amends original challenged claim 17 to further recite that “the plurality of engines are arranged in parallel with one another,” “each engine is configured to receive at least one of the plurality of tasks via a distinct command buffer for each engine,” “each of the plurality of tasks is decomposable into two or more related threads,” and “each engine is configured to load state data associated with at least one of the plurality of tasks.” Mot. App. A 2 (Claim Listing).

Petitioner asserts Wilt against claim 29. Thus, Petitioner relies on Wilt to teach or suggest that “the plurality of engines are arranged in parallel with one another,” “each engine is configured to receive at least one of the plurality of tasks via a distinct command buffer for each engine,” “each of the plurality of tasks is decomposable into two or more related threads,” and “each engine is configured to load state data associated with at least one of the plurality of tasks,” as recited in proposed substitute claim 29. We analyze Petitioner’s ground with respect to each of the limitations in proposed substitute claim 29 below.

amendment directed to “wherein the plurality of engines are arranged in parallel with one another” in claim 29

For this limitation, Petitioner relies on one of its proposed mappings in which “Wilt’s front end 212, work distribution unit 210, and core interface 303 of each PPU collectively serve as a single ‘engine,’ while cores 208 within and across all the PPUs collectively serve as the ‘shader core.’” Opp. 23. Petitioner argues Wilt teaches that “the plurality of engines are arranged in parallel with one another” (as required by proposed substitute claim 29) because

[Wilt’s] PPUs (PPU 0 ... PPU U-1) are arranged in parallel with one another, and the “one or more parallel processing units (PPUs) are part of the “parallel processing subsystem.” Ex. 1004, Fig. 2, 2:27–28, 3:36–48. Because there is a single “engine” within each PPU, the plurality of engines in Wilt are arranged in parallel.

Id.

Wilt appears to support Petitioner's contentions. Wilt discloses plural "parallel processing units (PPUs) 202, each of which is coupled to a local parallel processing (PP) memory 204. . . . a parallel processing subsystem includes a number U of PPUs, where $U \geq 1$," and each PPU includes a work distribution unit 210, a front end unit 212, and a core interface 303. *See* Ex. 1004, 3:37–40, Figs 2–3. There is, therefore, a work distribution unit 210, front end unit 212, and core interface 303 (collectively, an "engine") in each PPU, and the plurality of engines of the parallel PPUs are arranged in parallel with one another, as required by proposed substitute claim 29.

At this stage of the proceeding, it appears that Petitioner has sufficiently shown that Wilt teaches "wherein the plurality of engines are arranged in parallel with one another," as recited in proposed substitute claim 29.

We acknowledge that Patent Owner has not yet had the opportunity to address Petitioner's discussion of Wilt (and our analysis above) with respect to the claimed "parallel" engines limitation recited in proposed substitute claim 29. Patent Owner will have the opportunity to do so in a Reply to Petitioner's Opposition (or in a Revised Motion to Amend) in this proceeding. *See* Notice, *passim*.

amendment directed to "each engine is configured to receive at least one of the plurality of tasks via a distinct command buffer for each engine" in claim 29

Petitioner contends that "Wilt's host interface 206 in each PPU satisfies the claimed 'distinct command buffer' for each 'engine'" because

Within each PPU 202, the host interface 206 receives all incoming packets (the claimed "tasks") from the application software running on the CPU 102 (the claimed "scheduling module") via the communication path 113.[] Ex. 1004, 3:49–67. The host interface then sends the received tasks to the "engine" within each PPU (i.e., the front end unit 212, work distribution unit 210, and core interface 303 of each PPU). "For example, commands related to processing tasks may be directed to a front end unit 212." *Id.*, 3:60–61. Therefore, the host interface and corresponding "engine" in each PPU discloses or renders obvious the claimed "wherein each engine is configured to

receive at least one of the plurality of tasks via a distinct command buffer for each engine.”

Opp. 23–24.

Wilt appears to support Petitioner’s contentions. Wilt discloses that each PPU 202 includes a host interface 206. Ex. 1004, 3:49–51, Fig. 2. Host interface 206 of each PPU 202 reads commands for processing tasks received from a driver program on CPU 102, and each front end unit 212 of a PPU receives task commands from the host interface 206 of that PPU. *Id.* at 3:49–67, 4:61–5:51, Fig. 2. Work distribution unit 210 in the PPU receives commands defining tasks from front end unit 212 and distributes them to cores 208 via core interface 303, which receives data to process, parameters, and commands defining how to process the data (what program to execute) from work distribution unit 210, loads data into memory 306 and state parameters into memory 308 of core 208, and launches threads for execution in cores 208. *Id.* at 4:7–11, 8:14–21, 10:21–36.

In the Motion to Amend, Patent Owner presents arguments directed to the claimed “each engine is configured to receive at least one of the plurality of tasks via a distinct command buffer for each engine” recited in proposed substitute claim 29. Mot. 21. Patent Owner argues that “[u]nder either Wilt theory (the Single PPU Theory or the Multiple PPU Theory), the alleged engines in Wilt do not each have a distinct command buffer for each engine.” *Id.* Patent Owner reiterates that “[i]n the Multiple PPU Theory, the combination of those units [(front end 212, work distribution unit 210, and core interface 303)], one in each PPU, are not described as each having a distinct command buffer, much less one from which the alleged engines receive the alleged tasks.” *Id.* At this stage and based on the current record, we are unpersuaded by Patent Owner’s argument because, as discussed *supra*, it appears that Wilt’s host interface 206 in each PPU is “a distinct command buffer” for the “engine” of that PPU (front end 212, work distribution unit 210, and core interface 303) (as required by proposed substitute claim 29), and in each PPU, the “engine is configured to receive at least one of the plurality of tasks” via host interface 206/command buffer, as claimed.

Thus, at this stage of the proceeding, it appears that Petitioner has sufficiently shown that Wilt teaches “each engine is configured to receive

at least one of the plurality of tasks via a distinct command buffer for each engine,” as recited in proposed substitute claim 29.

We acknowledge that Patent Owner has not yet had the opportunity to address Petitioner’s discussion of Wilt with respect to the claimed “command buffer.” Patent Owner will have the opportunity to do so in a Reply to Petitioner’s Opposition (or in a Revised Motion to Amend) in this proceeding. *See Notice, passim.*

amendment directed to “each of the plurality of tasks is decomposable into two or more related threads” in claim 29

For this limitation recited in proposed substitute claim 29, Petitioner and Patent Owner rely on the arguments submitted with respect to the similar limitation recited in proposed substitute claim 28. *See* Opp. 24 (“*See* discussion above with respect to claim element 28[d].”); Mot. 21 (“As discussed above with respect to claim 28, Wilt’s alleged engines and the alleged shader core only receive and execute at best individual threads, and so the shader core does not simultaneously execute different tasks.”).

As discussed *supra* with respect to proposed substitute claim 28, at this stage of the proceeding, it appears that Petitioner has sufficiently shown that Wilt teaches or suggests processing of tasks where “each of the plurality of tasks is decomposable into two or more related threads.”

amendment directed to “each engine is configured to load state data associated with at least one of the plurality of tasks” in claim 29

Petitioner contends that Wilt teaches or renders obvious this added claim limitation in claim 29 because

Wilt uses the phrase “state parameters” to refer to state data. *Id.*; Ex. 1004 at 10:25–32. In Wilt, tasks are received by the host interface 206 in each PPU, which passes the tasks to the front end unit 212, then the work distribution unit 210, and finally, the core interface unit 303, which loads the *state parameters* (state data) associated with the task before passing the tasks to the cores 208 for processing. Paper 1, 27. “In some embodiments, core interface 303 receives data to be processed (e.g., primitive data, vertex data, and/or pixel data) as well as *state parameters* and commands defining how the data is to be processed (e.g., what

program is to be executed) from work distribution unit 210. Core interface 303 can *load* data to be processed into shared memory 306 and *[state] parameters* into parameter memory 308.” Ex. 1004, 8:15–21. Thus, where each “engine” is comprised of the front end unit 212, work distribution unit 210, and core interface unit 303 within each PPU, each “engine” is configured to load state data associated with at least one of the plurality of tasks it receives.

Opp. 25.

In the Motion to Amend, Patent Owner argues that Wilt does not teach or suggest this limitation recited in proposed substitute claim 29 because “[o]nly core interface 303 is alleged to load state data” while “[f]ront end unit 212 and work distribution unit 210 do not load state data.” Mot. 21 (citing Ex. 2007, 73:14–74:3; Ex. 2008 ¶ 74). At this stage and based on the current record, we are unpersuaded by Patent Owner’s argument because, as discussed *supra*, under Petitioner’s multiple PPU theory, *each engine* (front end 212, work distribution unit 210, and core interface 303 in a PPU) *is configured—via a portion of the engine* (the core interface, which loads state parameters into memory 308 of each core 208)—to load state data associated with at least one of the plurality of tasks, as required by proposed substitute claim 29.

Thus, at this stage of the proceeding, it appears that Petitioner has sufficiently shown that Wilt teaches or suggests “each engine is configured to load state data associated with at least one of the plurality of tasks,” as recited in proposed substitute claim 29.

We acknowledge that Patent Owner has not yet had the opportunity to address Petitioner’s discussion of Wilt with respect to the claimed loading of state data. Patent Owner will have the opportunity to do so in a Reply to Petitioner’s Opposition (or in a Revised Motion to Amend) in this proceeding. *See Notice, passim.*

In conclusion, based on the current record, at this stage of the proceeding, it appears that Petitioner has shown a reasonable likelihood that proposed substitute claim 29 is rendered obvious by Wilt.

III. CONCLUSION

This concludes the Preliminary Guidance, which is Patent Owner's first option under the Board's pilot program. Notice, 84 Fed. Reg. at 9,497. Patent Owner has an additional option under the Board's pilot program to file a revised MTA by DUE DATE 3. *See* Scheduling Order.

In addition to those two options under the Board's pilot program, Patent Owner is reminded that amendments of the challenged claims may be pursued in a separate reissue or reexamination proceeding before, during, or after an AIA trial proceeding, including subsequent to the issuance of the Final Written Decision. We draw Patent Owner's attention to the April 2019 *Notice Regarding Options for Amendments by Patent Owner Through Reissue or Reexamination During a Pending AIA Trial Proceeding*. *See* 84 Fed. Reg. 16,654 (Apr. 22, 2019). If Patent Owner chooses to file a reissue application or a request for reexamination of the challenged patent, we remind Patent Owner of its continuing obligation to notify the Board of any such related matters in updated mandatory notices. *See* 37 C.F.R. § 42.8(a)(3), (b)(2).

IPR2023-00687
Patent 8,854,381 B2

FOR PETITIONER:

Jeffrey Johnson
Jeffrey.johnson@bakerbotts.com

Christopher Higgins
0chptabdocket@orrick.com

Steven Baik
sbaik@whitehat.legal

FOR PATENT OWNER:

William Meunier
wameunier@mintz.com

Michael Renaud
mtrenaud@mintz.com

Adam Rizk
arizk@mintz.com

Michael Dokhanchy
rdokhanchy@mintz.com