IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re patent of Michael Rothschild	§	Attorney Docket No.: U022-0014RE
	§	
U.S. Patent 7,257,582	§	
	§	Customer No.: 29150
Issue Date: August 14, 2007	§	
	§	
Filing Date: February 27, 2003	§	
	§	
For: LOAD BALANCING WITH SHARED DATA	§	
	§	

Mail Stop "*Ex Parte* Reexam"
Attn: Central Reexamination Unit
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

REQUEST FOR *EX PARTE* REEXAMINATION OF U.S. PATENT 7,257,582

Dear Commissioner:

Pursuant to the provisions of 35 U.S.C. §§ 301-307, Unified Patents, LLC ("Requester") hereby requests an *ex parte* reexamination of claims 1-14 (the "Challenged Claims") of U.S. Patent No. 7,257,582 (the "'582 Patent," EX1001), which issued on August 14, 2007 to Michael Rothschild from U.S. Patent Application No. 10/375,893 (the "'893 Application"). The '893 Application was filed on February 27, 2003, claiming priority to Provisional Application No. 60/363,853 filed March 13, 2002. The '582 Patent is currently purportedly assigned to Intellectual Ventures I LLC. ("Intellectual Ventures" or "Patent Owner") according to an assignment recorded in the U.S. Patent and Trademark Office ("USPTO") at reel/frame 068714/0435.

Ex Parte Patent Reexamination Filing Requirements

Pursuant to 37 C.F.R. § 1.510(b)(l), statements pointing out at least one substantial new question of patentability (SNQ) based on material, non-cumulative reference patents and printed publications for the Challenged Claims of the '582 Patent are provided in Section V of this Request.

Pursuant to 37 C.F.R. § 1.510(b)(2), reexamination of the Challenged Claims of the '582 Patent is requested, and a detailed explanation of the pertinence and manner of applying the cited

references to the Challenged Claims is provided in Section VI of this Request. In addition, for each statement of the Patent Owner and accompanying information submitted pursuant to § 1.501(a)(2) that is relied upon in the detailed explanation, the Request explains how that statement is being used to determine the proper meaning of a patent claim in connection with the prior art applied to that claim and how each relevant claim is being interpreted.

Pursuant to 37 C.F.R. § 1.510(b)(3), copies of every patent or printed publication, relied upon or referred to in the statement pointing out each substantial new question of patentability or in the detailed explanation of the pertinence and manner of applying the cited references, are provided as Exhibits 1005 (Iyer) and 1006 (Reeve) of this Request. Requester also includes an Information Disclosure Statement listing all documents for consideration by the Office.

Pursuant to 37 C.F.R. § 1.510(b)(4), a copy of the '582 Patent is provided as Exhibit 1001 of this Request, along with a copy of any disclaimer, certificate of correction, and reexamination certificate issued corresponding to the patent.

Pursuant to 37 C.F.R. § 1.510(b)(5), the attached Certificate of Service indicates that a copy of this Request, in its entirety, has been served on Patent Owner at the following address of record for Patent Owner, in accordance with 37 C.F.R. § 1.33(c):

86636 - BRUNDIDGE & STANGER, P.C.
1925 BALLENGER AVENUE, STE. 560
ALEXANDRIA, VA 22314
UNITED STATES

In addition, as a courtesy, Requester has served a courtesy copy of the Request by email on Patent Owner's litigation counsel as follows:

Jonathan K. Waldrop
jwaldrop@kasowitz.com
Darcy L. Jones
djones@kasowitz.com
Marcus A. Barber
mbarber@kasowitz.com
John W. Downing
jdowning@kasowitz.com
Heather S. Kim
hkim@kasowitz.com
ThucMinh Nguyen
tnguyen@kasowitz.com
KASOWITZ BENSON TORRES LLP
333 Twin Dolphin Drive, Suite 200
Redwood Shores, California 94065

Pursuant to 37 C.F.R. § 1.510(b)(6), Requester hereby certifies that the statutory estoppel provisions of 35 U.S.C. § 315(e)(1) and 35 U.S.C. § 325(e)(1) do not prohibit Requester from filing this *ex parte* patent reexamination request.

Also submitted herewith is the fee set forth in 37 C.F.R. § 1.20(c)(l).

Table of Contents

I. The '582 Patent 8

A. Priority 8

B. Overview 8

C. Prosecution History 9

1. Original Prosecution 9

2. IPR2025-00785, *American Airlines, Inc. et al v. Intellectual Ventures I LLC* . 11

D. Challenged Claims 11

E. Level of Ordinary Skill in the Art 13

II. Claim Construction 13

III. Listing of Prior art patents and printed publications 13

IV. Overview of the prior art 14

A. Background and State of the Art 14

B. Iyer (EX1005) 16

C. Reeve (EX1006) 16

D. Analogous Art 18

V. Substantial New Question of Patentability 19

VI. Detailed Application of the Prior Art to Every Claim for Which Reexamination is Requested 22

A. Ground 1: Iyer and Reeve Render Obvious Claims 1-14 22

1. Independent Claim 1 22

2. Dependent Claim 2 37

3. Dependent Claim 3 38

4. Dependent Claim 4 39

5. Dependent Claim 5 40

6. Dependent Claims 6 and 7 40

7. Dependent Claims 8-11 43

8. Dependent Claim 12 44
9. Dependent Claim 13 45
10. Dependent Claim 14 46

B. Grounds 2 and 3: Reeve Anticipates and/or Renders Obvious Claims 1-14 47

1. Independent Claim 1 47
2. Dependent Claim 2 58
3. Dependent Claims 3-5 60
4. Dependent Claims 6 and 7 61
5. Dependent Claims 8-11 62
6. Dependent Claim 12 63
7. Dependent Claim 13 64
8. Dependent Claim 14 66

VII. Disclosure of Concurrent litigation, Reexamination, and Related Proceedings 67
VIII. Conclusion 68

TABLE OF EXHIBITS

Exhibit No.	Description of Document
1001	U.S. Patent 7,257,582 B2 ("'582 Patent") to Rothschild filed February 27, 2003 and issued August 14, 2007)
1002	File history for the '582 Patent
1003	Declaration of Dr. Jon B. Weissman
1004	*Curriculum Vitae* of Dr. Jon B. Weissman
1005	U.S. Patent No. 5,179,699 to Iyer et al. ("Iyer")
1006	U.S. Patent No. 5,535,393 to Reeve et al. ("Reeve")
1007	Petition for *Inter Partes* Review in *American Airlines, Inc. et al v. Intellectual Ventures I LLC*, IPR2025-00785.
1008	Excerpt of Microsoft Computer Dictionary
1009	Wikipedia Pages – Storage Area Network, Network-attached Storage
1010	"Some Thoughts on Parallel Processing," Lynn D. Yarbrough, Communications of the ACM, Vol. 3, No. 10, Oct. 1960
1011	"A Survey of Some Theoretical Aspects of Multiprocessing," J. L. Baer, ACM Computing Surveys, Vol. 5, No. 1, Mar. 1973
1012	"Parallel Sorting Algorithms for Tightly Coupled Multiprocessors," Michael J. Quinn, Parallel Computing, Vol. 6, 1988
1013	"Removing Skew Effect in Join Operation on Parallel Processors," Ron-Chung Hu et al., Univ. of California, Los Angeles, Jun. 1989
1014	"Parallel Architectures for Database Systems," A.R. Hurson et al., Advances in Computers, Vol. 28, 1989
1015	"Parallel Algorithms for Merging and Sorting," Narsingh Deo, Dilip Sarkar, Information Sciences, Vol. 56, 1991
1016	"Parallel Algorithms for Least Median of Squares Regression," Chong-Wei Xu, Wei-Kei Shiue, Computational Statistics & Data Analysis, Vol. 16, 1993
1017	"On the Average Running Time of Odd-Even Merge Sort," Christine Rüb, Max Planck Institute Technical Report, Apr. 1995
1018	"An Approach to Parallelizing Isotonic Regression," Anthony J. Kearsley et al., Applied Mathematics and Parallel Computing, 1996
1019	"Parallel Algorithms," Guy E. Blelloch, Bruce M. Maggs, in The Computer Science and Engineering Handbook, CRC Press, 1997
1020	"Industrial Strength Parallel Computing," Alice E. Koniges et al., Morgan Kaufmann, 2000

1021	"Analysis of First-Come-First-Serve Parallel Job Scheduling," Uwe Schwiegelshohn, Ramin Yahyapour, SODA, 1998
1022	"GPFS: A Parallel File System," Jason Barkes et al., IBM Technical Report SG24-5165-00, Apr. 1998
1023	U.S. Patent No. 6,304,866 to Chow et al., issued Oct. 16, 2001 ("Chow")
1024	U.S. Patent No. 6,330,583 to Reiffin, issued Dec. 11, 2001 ("Reiffin")
1025	U.S. Patent App. Pub. No. 2002/0019844 to Kurowski et al., published Feb. 14, 2002 ("Kurowski")
1026	"Impact of Self-Scheduling Order on Performance of Multiprocessor Systems," Peiyi Tang et al., Proceedings of the 2nd International Conference on Supercomputing, 1988

I. THE '582 PATENT

A. Priority

The '582 Patent was filed on February 27, 2003, as U.S. Application No. 10/375,893 (the "'893 Application"). The '582 Patent claims priority to Provisional Application No. 60/363,853 ("the '853 Provisional") filed March 13, 2002. Thus, pre-AIA 35 U.S.C. § 103 applies. All of Requester's prior art predates both the provisional and non-provisional filing dates of the '582 Patent. Thus, for purposes of this reexamination, it does not matter whether the '582 Patent is entitled to its earliest claimed priority date.

B. Overview

The '582 Patent relates to conventional load balancing using multiple computer processors—i.e., a "multiprocessor." EX1001, 1:14-17, 2:37 Specifically, the '582 Patent describes a long-known "way to split the performance of a given task among a plurality of processing units which can all access, directly or indirectly, the input data and the devices on which the output data is to be stored." *Id.*; *see also id.*, 1:48-54 ("The principal object of the present invention is to enable the decomposition of a certain type of linear processes that currently use a single computer, into equivalent parallel processes that can efficiently use any number of potentially heterogeneous computers, taking the available capacity of each of these computers into account while optimizing execution.").

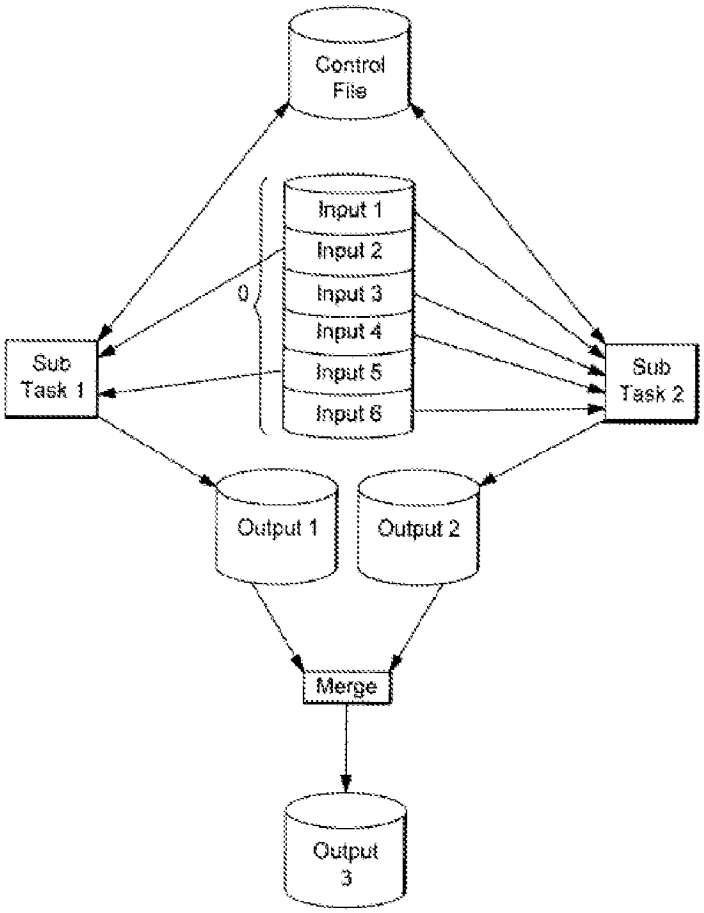


Fig. 4

Figure 4 of the '582 Patent, reproduced above, shows two processors, subtask 1 and subtask 2, cooperating to sort an "input file 0 [that] is (logically) partitioned into six partitions." *Id.*, 6:14-24, FIG. 4. In this sort process:

- Sub Task 2 updates the Control File to claim Input 1 and begins processing it.
- While Sub Task 2 processes Input 1, Sub Task 1 claims Input 2 and starts processing it.
- Sub Task 2 happens to be faster than Sub Task 1 (e.g., faster processor or lower load). Thus, Sub Task 2 finishes Input 1 before Sub Task 1, claims Input 3, and starts processing it. Sub Task 2 also finishes Input 3 before Sub Task 1 finishes Input 2, so Sub Task 2 claims and processes Input 4.
- In the meantime, Sub Task 1 finishes Input 2, claims Input 5, and processes it.
- The processes continues in this way until all Inputs are processed.
- Finally, "[w]hen both Sub Tasks terminate the processing of their last input partition they put the sorted output on their respective output files, and the system "merges the two output files into one merged output file."

Id., 6:19-42. In short, the Sub Task processors sequentially process "each ... unprocessed partition on a first-come/first-served basis," as recited in '582 Patent claim 1, until the whole file is processed. *Id.*, claim 1.

C. Prosecution History

1. Original Prosecution

In the brief prosecution history of the '582 Patent, the Applicant focused on the above-described "first-come/first-served" aspect of the '582 Patent's process to distinguish claim 1 over cited prior art. Specifically, in a Non-final Office Action, the Examiner rejected the claims of the '893 Application as having been obvious over U.S. Patent 5,603,028 ("Kitsuregawa") in view of U.S. Patent 5,357,632 ("Pian"). EX1002, 56-60.

In response, the Applicant amended independent claim 1 to, among other things, add details of the "first-come/first-served" load balancing process:

1. (currently amended) A method of effecting on a preexisting input file a computer-executable process comprised of a plurality of subtasks, the method comprising the steps of:

(a) automatically determining file allocation and logically subdividing records of said input file into a plurality of partitions;

(b) distributing descriptions of all of said partitions to each of a plurality of subtask processors ~~and activating~~

(c) simultaneously executing at least a respective one of the subtasks of the computer-executable process in each of at least some of said processors on a respective one of the partitions with [[,]] each subtask reading and processing [[said]] the respective partition [[s]] so as to process the respective partition and produce respective subtask output and;

(d) thereafter repeating step (c) in at least some of the subtask processors each with another unprocessed partition on a first-come/first-served basis; and

[[(c)]] (e) generating at least one output combining all of the subtask outputs and reflecting the processing of all of said subtasks.

Id., 47.

In the remarks, the Applicant provided context on the load sharing process and explained how it purportedly differs from those of Kitsuregawa and Pian. *See id.*, 51-52. Particularly, the Applicant contended that, unlike the cited prior art, the claimed technique did not have any "special process" overseeing the load balancing process that manages distribution of the tasks to the processors:

> The primary difference between the instant invention and the processes disclosed in US 5,603,028 of Kitsuregawa and 5,357,632 of Pian is that these systems rely on a special control process that uses load information to distribute the load between processors that share the load. With the instant invention as defined in the claims there is no such special process. The prior art's load information is not created with the process of the instant invention. Instead, the load sharing is done as a byproduct of the fact that the load-sharing process take parts of the load on a first-come/first-served basis.

> A comparison would be to a road intersection where, according to the prior art, there is a traffic light that determines who can go when. The instant invention is more like such an intersection with a four-way stop so that the individual drivers determine who can go and when.
>
> This is a major improvement since in addition to eliminating the control process-it also eliminates the need to collect and maintain load information, which it is very difficult. to do and almost impossible to define so as to anticipate all possible processors that might execute the subtasks.
>
> The amended claims, refer to a distribution of a description of the work to be done. The sharing process can use such a description to distribute the load without a special load process.

Id., 51-52. Following Applicant's response, the Examiner allowed the '893 Application, without a separate statement on the Examiner's reasons for allowance. *Id.*, 39-45.

2. IPR2025-00785, *American Airlines, Inc. et al v. Intellectual Ventures I LLC*

The '582 Patent is the subject of an *inter partes* review petition filed on April 10, 2025 in *American Airlines, Inc. et al v. Intellectual Ventures I LLC*, IPR2025-00785. *See* EX1007. The PTAB has not yet assessed the merits of the IPR petition, and a Decision on Institution is not due until November 10, 2025. Paper 4 (Notice Granting Filing Date issued May 9, 2025; November 9, 2025 being a Sunday). This Request relies on none of the prior art applied in the *American Airlines* IPR Petition. *See id.* (relying on Chow (EX1023), Reiffin (EX1024), and Kurowski (EX1025)).

D. Challenged Claims

Requester reproduces Challenged Claims 1-14 below with claim element reference numbers in brackets used in Section VI.

> 1. [1.pre] A method of effecting on a preexisting input file a computer-executable process comprised of a plurality of subtasks, the method comprising the steps of:
>
> [1.1] (a) automatically determining file allocation and logically subdividing records of said input file into a plurality of partitions;
>
> [1.2] (b) distributing descriptions of all of said partitions to each of a plurality of subtask processors
>
> [1.3] c) simultaneously executing at least a respective one of the subtasks of the computer-executable process in each of at least some of said processors on a respective one of the partitions with each subtask reading and processing the respective partition so as to process the respective partition and produce respective subtask output and;
>
> [1.4] d) thereafter repeating step (c) in at least some of the subtask

processors each with another unprocessed partition on a first-come/first-served basis; and

[1.5] (e) generating at least one output combining all of the subtask outputs and reflecting the processing of all of said subtasks.

2. [2.1] The method defined in claim 1 wherein the automatic determination of file allocation and logical subdivision of records of said input file into said plurality of partitions in step (a) and the distribution of the description of all of said partitions in step (b) is carried out with at least one further processor in addition to the subtask processors.

3. [3.1] The method defined in claim 1, further comprising the step of merging said subtask outputs in step (e).

4. [4.1] The method defined in claim 1 wherein the output in step (e) is a succession of outputs from said subtasks in a one-to-one correspondence with said records of said input file.

5. [5.1] The method defined in claim 1 wherein the output in step (e) is an accumulation of output records from said subtasks in an arbitrary order.

6. [6.1] The method defined in claim 1 wherein said input file resides on a storage area network and is derived therefrom.

7. [7.1] The method defined in claim 1 wherein said input file resides on a network-attached storage and is derived therefrom.

8. [8.1] The method defined in claim 1 wherein said computer-executable process is a sort process.

9. [9.1] The method defined in claim 1 wherein said computer-executable process is a statistical analysis process.

10. [10.1] The method defined in claim 1 wherein said computer-executable process is a report-creating process.

11. [11.1] The method defined in claim 1 wherein said computer-executable process includes a database query.

12. [12.1] The method defined in claim 2 wherein said one processor is part of a mainframe computer and the other processors are processors of at least one other computer.

13. [13.1] The method defined in claim 1 wherein said plurality of processors are all parts of a single multiprocessor.

14. [14.1] The method defined in claim 1 wherein the automatic determination of file allocation and logical subdivision of records of said input file into said plurality of partitions in step (a) and the distribution of the descriptions of all of said partitions in step (b) is carried out with at least one processor, and said one processor and said plurality of processors are all parts of a single multiprocessor not including said one processor.

E. Level of Ordinary Skill in the Art

A person of ordinary skill in the art ("POSITA") at the time of the '582 Patent's claimed priority date would have had a bachelor's degree in computer science, computer engineering, electrical engineering, or related field, and at least two years of experience in computer networking and parallel computing or have had a master's degree in one of the foregoing and at least one year of experience in the aforementioned fields. Additional education could substitute for professional experience, and *vice versa*. EX1003, ¶ 43.

II. CLAIM CONSTRUCTION

Because the '582 Patent does not expire until August 2025, the Challenged Claims at this time should be given their broadest reasonable interpretation. MPEP § 2258.I.G (citing *In re Yamamoto*, 740 F.2d 1569, 222 USPQ 934 (Fed. Cir. 1984)); 37 C.F.R. § 1.555.[1] Under this standard, "[t]he Patent and Trademark Office … determines the scope of claims in patent applications not solely on the basis of the claim language, but upon giving claims their broadest reasonable construction in light of the specification as it would be interpreted by one of ordinary skill in the art. " *Phillips v. AWH Corp.*, 415 F.3d 1303, 1316, 75 USPQ2d 1321, 1329 (Fed. Cir. 2005) (quotation and citation omitted). "Because applicant has the opportunity to amend the claims during prosecution, giving a claim its broadest reasonable interpretation will reduce the possibility that the claim, once issued, will be interpreted more broadly than is justified." MPEP § 2111 (citing *In re Yamamoto*, 740 F.2d 1569, 1571 (Fed. Cir. 1984). Requester does not believe that any terms of the Challenged Claims requires express construction. Although Requester applies the broadest reasonable construction, Requester's prior art renders the Challenged Claims unpatentable under any reasonable construction, including under the plain and ordinary meaning standard according to *Phillips*.

III. LISTING OF PRIOR ART PATENTS AND PRINTED PUBLICATIONS

Requester requests reexamination of the Challenged Claims in view of the following documents. The chart also indicates each document's prior-art date and the statutory basis for being prior art, with March 13, 2002 being the presumed effective filing date for the Challenged Claims.

---

[1] In the event that the '582 Patent expires during the reexamination proceeding, the *Philips* standard for claim construction will apply. *In re CSB-Sys. Int'l, Inc.*, 832 F.3d 1335, 1341 (Fed. Cir. 2016). No matter which standard applies, Requester's prior art satisfies the claim elements.

Exhibit	*Prior Art Document*	*Prior Art Date*	*Statutory Basis*[2]
EX1005	U.S. Patent No. 5,179,699 ("Iyer)	Jan. 13, 1989 Jan 12, 1993	35 U.S.C. § 102(e) 35 U.S.C. §§ 102(a), (b)
EX1006	U.S. Patent No. 5,535,393 ("Reeve")	Jun. 5, 1995 Jul. 9, 1996	35 U.S.C. § 102(e) 35 U.S.C. §§ 102(a), (b)

Iyer is a U.S. Patent entitled, "Partitioning of sorted lists for multiprocessors sort and merge," filed January 13, 1989 and issued January 12, 1993. EX1005, 1. Iyer is prior art under at least 35 U.S.C. §§ 102(a), (b), and (e).

Reeve is a U.S. Patent entitled, "System for parallel processing that compiles a filed sequence of instructions within an iteration space," filed June 5, 1995 and issued July 9, 1996. EX1006, 1. Reeve is prior art under at least 35 U.S.C. §§ 102(a), (b), and (e).

Neither Iyer nor Reeve was cited during prosecution of the '582 Patent or in the pending IPR Petition filed by American Airlines. *See* EX1001, 1; EX1007. A Form SB-08 and copies of the cited references are submitted with this Request. As shown below, Requester submits that the prior art references raise multiple substantial new questions of patentability of at least one of the Challenged Claims.

IV. OVERVIEW OF THE PRIOR ART

The elements of the load-balancing technique described in the '582 Patent, and the overall technique itself, were known long before the '582 Patent's earliest claimed priority date. EX1003, ¶¶ 30-35.

A. Background and State of the Art

As Dr. Weissman explains, as of the claimed March 13, 2002 priority date of the '582 Patent, the fundamental principles of parallel processing were well-established and extensively documented in the literature. EX1003, ¶¶ 30-35. For decades, practitioners in the field had understood how to divide computational tasks into subtasks, assign them to multiple processors and recombine their outputs to produce a final result.

[2] Pre-AIA provisions govern this proceeding.

Techniques such as dynamic load balancing, first-come/first-served scheduling, partitioning of input data, and coordination by a central or supervisory processor were all known. The prior art also disclosed applying these methods to a variety of specific applications, including sorting, database queries, statistical analysis, and report generation. The '582 Patent repackages these longstanding practices without contributing any technical advancement. Notably, no prior art was submitted during prosecution, and the patent issued after minimal examination. In my opinion, every claim of the '582 Patent reflects conventional techniques that would have been familiar to a person of ordinary skill in the art well before the priority date. EX1003, ¶ 30-34.

The following references, published between the 1960s and early 2000s, illustrate the well-established state of the art in parallel computing as of the '582 Patent's priority date. Each reference independently discloses one or more limitations of the challenged claims, including key features such as partitioning input data, distributing tasks to multiple processors, dynamic scheduling, and merging outputs. Collectively, they confirm that the claimed subject matter reflects routine and predictable variations on techniques already known to those of ordinary skill in the art. A brief summary of each reference is provided below. *Id.*, ¶ 35.

- Iyer et al. (1989) (EX1005) – Discloses partitioning and merging of sorted lists across multiple processors with efficient load balancing among processors during sorting and merging, directly relevant to claim 1 and the elements involving sorting and final output combination.
- Reeve et al. (1995) (EX1006) – Describes a tiled parallel execution strategy where multiple processors execute tiles of an iteration space and dynamically "grab" new tiles when idle, with the combined result producing the same output as a serial execution—disclosing subtask distribution, dynamic load balancing, and output merging.
- Yarbrough (1960) (EX1010) – Discloses a system with multiple processors and a monitor processor accessing shared memory for parallel execution—corresponding to partitioning, task distribution, and coordination found in the claims.
- Baer (1973) (EX1011) – Surveys parallel processing architectures and explicitly discusses parallel sorting, aligning with the sort process of claim 8.
- Quinn (1988) (EX1012) – Describes and compares various parallel sorting algorithms, such as quickmerge and parallel Shellsort, covering claims 3 and 8.
- Hu et al. (1989) (EX1013) – Teaches parallel execution of database join queries with final result merging, matching the database query and merging steps in claims 3 and 11.
- Hurson et al. (1989) (EX1014) – Categorizes parallel architectures (e.g., MIMD, SIMD) and identifies configurations involving mainframes and distributed systems, relevant to claims 12 and 13.
- Deo et al. (1991) (EX1015) – Introduces parallel algorithms for merging and sorting by partitioning datasets, directly mapping to claims 3 and 8.

- Xu et al. (1993) (EX1016) – Applies parallel algorithms to least median of squares statistical regression, addressing statistical analysis in claim 9.
- Rüb (1995) (EX1017) – Analyzes average running time for parallel odd-even merge sort using multiple processors, relevant to claims 3 and 8.
- Kearsley et al. (1996) (EX1018) – Describes parallel isotonic regression on subsets of data, aligning with statistical analysis in claim 9.
- Blelloch et al. (1997) (EX1019) – Details memory models and load balancing techniques in multiprocessors, disclosing storage area and network-attached storage (claims 6 and 7).
- Koniges et al. (2000) (EX1020) – Explains industrial parallel architectures including use of master processors and heterogeneous clusters, supporting claims 2, 12, and 13.
- Schwiegelshohn et al. (1998) (EX1021) – Analyzes First-Come-First-Serve scheduling in parallel systems, directly mapping to claim 1(d)'s scheduling limitation.
- Barkes et al. (1998) (EX1022) – Describes GPFS parallel file system for sorting and reporting, corresponding to claims 8 and 10.
- Chow et al. (2001) (EX1023) – Teaches dynamic, on-demand task subdivision and load-balanced execution with subtask merging, directly forming the foundation for multiple challenged claims.
- Reiffin (EX1024) – Provides a distributed task model where networked machines asynchronously pull subtasks from a shared queue, supporting partition distribution and first-come, first-served (FCFS) processing in claim 1.
- Kurowski et al. (2002) (EX1025) – Details a large-scale distributed computing system with servers managing subtask distribution to heterogeneous clients, relevant to Grounds 2 and claims involving distributed execution.
- Tang et al. (1988) (EX1026) – Describing a self-scheduling (another name for FCFS) strategy in which processors dynamically request new tasks as they become idle—effectively assigning subtasks on a first-come, first-served basis to maximize utilization and balance load.

EX1003, ¶ 35.

B. Iyer (EX1005)

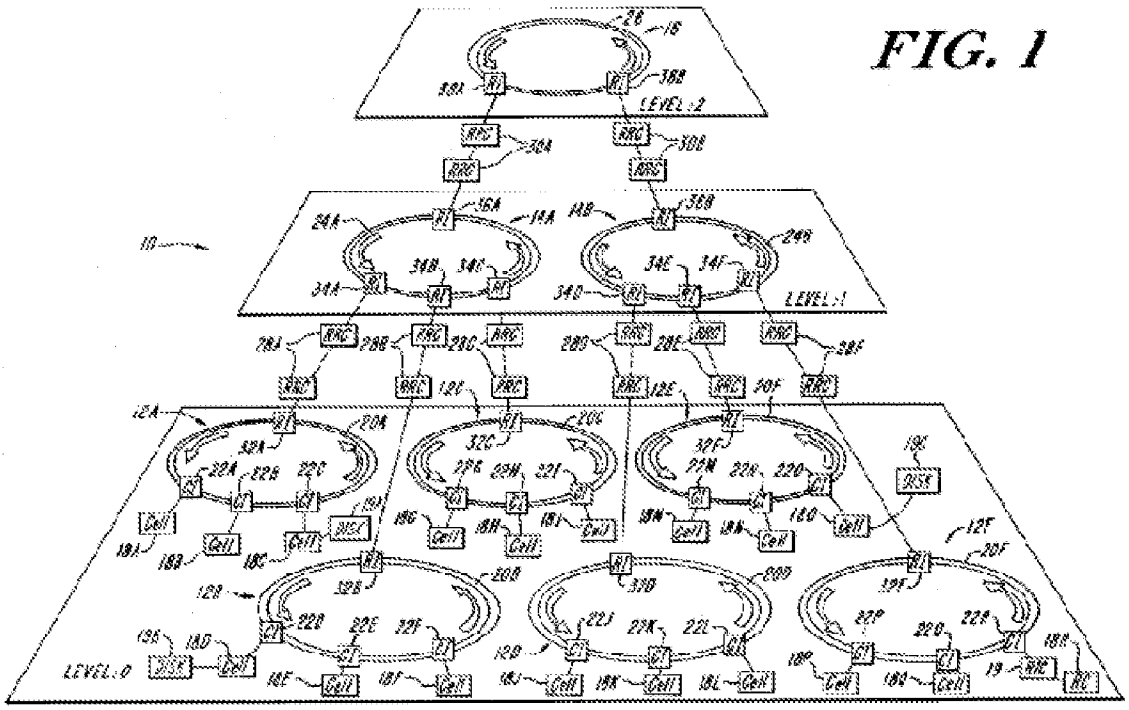
Iyer describes a "multiprocessor computer system" configured to sort and merge a list. EX1005, 3:55-61; *see also id.*, 1 (title). The system includes processors 1-P, and processor 1 divides an initial list into P partitions—one for each processor 1-P. *Id.*, Abstract, 2:21-3:50, 4:21-58, 10:66-11:41. The processors 1-P sort and merge their partitions, which processor 1 concatenates into a final, sorted list. *Id.*, Abstract, 2:33-51, 10:66-11:41, FIGS. 3A, 3B.

C. Reeve (EX1006)

Like the '582 Patent, Reeve describes a "multiprocessor" load-balancing technique for parallelizing—in Reeve's case—execution of iterative sequences in a program. *See* EX1006, Abstract, 1:31-55, Reeve's technique, called "tiling", partitions loops and other iterative sequences

of the program into tiles (subtasks), which are allocated to multiple parallel processors for execution. *Id.*, 2:3-4:59.

Figure 1 of Reeve, reproduced below, "depicts a preferred multiprocessing system [10] ... [that] includes three information transfer levels: level:0, level:1, and level:2." *Id.*, 5:23-26.



Reeve, FIG. 1

The bottom level, level 0, contains Reeve's multiprocessor cells that execute the tiled sequences:

> Particularly, level:0 of the illustrated system 10 includes six segments, designated 12A, 12B, 12C, 12D, 12E and 12F, respectively. Similarly, level:1 includes segments 14A and 14B, while level:2 includes segment 16.
>
> Each segment of level:0, i.e., segments 12A, 12B, . . . 12F, comprise a plurality of processing cells. For example, segment 12A includes cells 18A, 18B and 18C; segment 12B includes cells 18D, 18E and 18F; and so forth. Each of those cells include a central processing unit and a memory element, interconnected along an intracellular processor bus (not shown).

EX1006, 5:28-32.

Shown in Figure 3, reproduced on the right, Reeve describes "[a] compilation system 60 [that] translates source code input into object code. The source code can be of conventional format, e.g., Fortran 77 or Fortran 90, or C programming language source files, and typically includes iterative sequences or 'loops.' EX1006, 7:14-21.

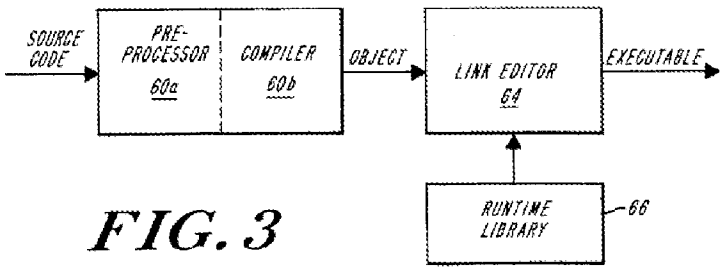


FIG. 3

Compilation system 60 includes a "preprocessor 60a [that] performs preliminary analysis of iterative sequences in the source code to determine the dependency directions thereof, and performs certain loop interchanges," and "generates [tiling] directives " in a tiling process that partitions the loops of the program. *Id.*, 7:28-47. "Those directives are preferably provided as 'comments,' i.e., non-executable statements." *Id.*, 7:23-27. In short, "[t]he main role of preprocessor 60a is to put annotations into [the] Fortran program that enable parallel execution; that is, to insert tiling directives." *Id.*, 8:1-3. Reeve teaches an "automatic tiling" process in which preprocessor 60a does this process in a "fully-automatic" manner. *Id.*, 9:37-44.

Preprocessor 60a outputs "[a] second program [that] is a valid and correct Fortran program, which when executed serially computes the same results as the original program. That second program is the input to the compiler 60b." *Id.*, 8:8-12. Based on the second program, "compiler 60b ... translates program statements from the preprocessed source code format to object code format. In addition to translating conventional code, the compiler 60b converts iterative sequences in the preprocessed source code to 'tiled' sequences for use in parallel execution." *Id.*, 7:38-43.

D. Analogous Art

Iyer and Reeve are analogous art to the '582 Patent and to each other. "A reference is analogous art to the claimed invention if: (1) the reference is from the same field of endeavor as the claimed invention (even if it addresses a different problem); or (2) the reference is reasonably pertinent to the problem faced by the inventor (even if it is not in the same field of endeavor as the claimed invention)." MPEP § 2141.01(a) (citing *In re Bigio*, 381 F.3d 1320, 1325 (Fed. Cir. 2004)). Iyer, Reeve, and the '582 Patent are in a same field of endeavor—multiprocessor systems. EX1001, Abstract, 1:14-17, 2:37; EX1005, 3:55-61; *see also id.*, 1 (title); EX1006, Abstract, 1:32-

38, 2:65-67, 4:66-67, 5:23-32; EX1003, ¶ 53.

In addition, Reeve and Iyer are reasonably pertinent to a problem faced by the purported inventors of the subject matter recited in the Challenged Claims of the '582 Patent. EX1003, ¶ 54. The '582 Patent observes that "many computer sites contain a plurality of computers," but the computers often have uneven workloads, with "one of them [being] especially loaded while others are relatively idle." EX1001, 1:25-33. To address this imbalance, the '582 Patent aims to use "parallelization of some processes and their distributed execution across all available computing resources in a way that both speeds the execution of the whole process and balances the load the various computers are subjected to." *Id.*, 1:33-38. Reeve is directed to the same type of problem, noting that "'[p]arallelizing' execution of computer programs so that they can be run efficiently on parallel processors is a daunting task." EX1006, 1:39-41. Similarly, Iyer aims to overcome a problem in parallel sorting algorithms where "partitions can be greatly differing in size thereby exacerbating the load imbalance problem." EX1005, 8-13.

V. SUBSTANTIAL NEW QUESTION OF PATENTABILITY

Requester requests reexamination of the Challenged Claims based on Grounds 1-3 in the table below, each corresponding to a substantial new question of patentability:

Challenge	Claims	Statutory Basis	Prior Art References
Ground 1	1-14	35 U.S.C. § 103	• Iyer (EX1005) • Reeve (EX1006)
Ground 2	1-14	35 U.S.C. § 102	• Reeve (EX1006)
Ground 3	1-14	35 U.S.C. § 103	• Reeve (EX1006)

For each of Grounds 1-3, a substantial new question of patentability is based on Reeve alone, as a primary reference, or as a secondary reference in combination with Iyer. This Request is also supported by a declaration of its technical expert, Dr. Jon Weismann. EX1003.

Reeve presents a new technological teaching of element [1.4] recited in independent claim 1 and required by all Challenged Claims: "*d) thereafter repeating step (c) in at least some of the subtask processors each with another unprocessed partition* ***on a first-come/first-served basis.***"[3] As explained above, the Examiner allowed the '582 Patent claims based on Applicant's argument

[3] All emphasis is added unless otherwise specified.

that the claimed process differs from the prior art because the processors take parts of the load on a "first-come, first-serve" basis, without requiring a special load-balancing process to distribute the load to the processors:

> The primary difference between the instant invention and the processes disclosed in US 5,603,028 of Kitsuregawa and 5,357,632 of Pian is that these systems rely on a special control process that uses load information to distribute the load between processors that share the load. With the instant invention as defined in the claims there is no such special process. The prior art's load information is not created with the process of the instant invention. Instead, the load sharing is done as a byproduct of the fact that the load-sharing process take parts of the load on a first-come/first-served basis.
>
> A comparison would be to a road intersection where, according to the prior art, there is a traffic light that determines who can go when. The instant invention is more like such an intersection with a four-way stop so that the individual drivers determine who can go and when.
>
> This is a major improvement since in addition to eliminating the control process-it also eliminates the need to collect and maintain load information, which it is very difficult. to do and almost impossible to define so as to anticipate all possible processors that might execute the subtasks.
>
> The amended claims, refer to a distribution of a description of the work to be done. The sharing process can use such a description to distribute the load without a special load process.

Id., 51-52; *see also id.*, 39-45 (no separate Examiner's reasons for allowance).

Reeve provides a new technical teaching regarding element [1.4] for all Challenged Claims. In particular, Reeve discloses a multiprocessor load balancing system that employs a "GRAB strategy" in which cell processors take work on a "first-come, first-serve" basis:

> A grab strategy too divides the iteration space such that there are more tiles than available processors. The resultant tiles are assigned to requesting processors on a first-come-first-serve basis.

EX1006, 4:1-7.

> When using the GRAB strategy, Runtime Library 66 lets the scheduler handle all thread-processor bindings, load balancing, affinity, etc.

Id., 22:56-58.

> GRAB strategy
>
> This strategy issues tiles on a first-come, first-serve basis to the thread group. Each

> thread must obtain a common lock to access a tile for execution. This strategy is designed to load balance between the threads of the thread group.
>
> Note that this strategy does not consider data affinity considerations when assigning tiles to processors, and that it would not be a good strategy to choose in conjunction with affinity regions.

Id., 26:14-22.

> The GRAB tiling strategy, is dynamic and will keep each thread busy as much as possible, but will probably cause data migration to occur more frequently. In addition to affinity loss problems is the additional synchronization overhead from the locking required to access a new tile.

Id., 26:32-36.

> 6.1.2 Locking during the grab strategy (critical section)
>
> The grab strategy requires all member threads to update a marker data structure to get the next available tile. This marker data structure is held in a lock.

Id., 42:28-31.

Reeve and Iyer were not before the Examiner during original prosecution—nor was any cumulative prior art. Had the Examiner been aware of Reeve and considered its teachings regarding the GRAB strategy for processing records on a first-come, first-serve basis, the Examiner would have rejected Challenged Claims 1-14. Thus, the "same question of patentability as to the claim has not been decided by the Office in an earlier concluded examination or review of the patent." MPEP § 2642.

While the '582 Patent is the subject of an IPR Petition in IPR2025-00785, Requester relies on none of the same references or grounds as that Petition. EX1007. Moreover, as explained, an Institution Decision is not expected until November 2025. The PTAB has not decided the merits of the Petition, and would not do so until late 2026, assuming trial is instituted and proceeds to a Final Written Decision.[4] Thus, the same question of patentability has not, and will not, be decided in the IPR. If necessary, the PTAB has discretion to "determine the manner in which the post-grant

[4] In addition to the Request's presenting different prior art and arguments than the IPR Petition, discretionary denial under § 325(d) would be inappropriate because the PTAB has not yet considered the merits of the Petition, so the prior art and arguments have not been "presented to the Office." 35 U.S.C. § 325(d).

review or other proceeding or matter may proceed," such as to stay this Request pending resolution of the *American Airlines* IPR Petition. 35 U.S.C. § 325(d).

VI. DETAILED APPLICATION OF THE PRIOR ART TO EVERY CLAIM FOR WHICH REEXAMINATION IS REQUESTED

In Grounds 1-3 below, Requester explains in detail how the prior art references listed above establish the unpatentability of the Challenged Claims as either anticipated under 35 U.S.C. § 102, obvious under 35 U.S.C. § 103, or both. Requester respectively requests the Office to issue an Office Action rejecting the Challenged Claims based on each of Grounds 1-3. This Request is supported by a declaration of Requester's technical expert, Dr. John Weissman. *See* EX1003.

A. Ground 1: Iyer and Reeve Render Obvious Claims 1-14

1. Independent Claim 1

a. [1.pre]: "*A method of effecting on a preexisting input file a computer-executable process comprised of a plurality of subtasks, the method comprising the steps of:*"

To the extent the preamble is found to be limiting, Iyer discloses *a method* (multiprocessor sort and merge) *of effecting on a preexisting input file* (original large list) *a computer-executable process* (sort and merge) *comprised of a plurality of subtasks* (processors 1-P sorting partitions of the original large list), as claimed. EX1003, ¶¶ 64-66.

Regarding the *computer-executable process*, Iyer discloses:

> Any number of sorted lists are efficiently partitioned into P lists, where P represents the number of processors available to sort the resulting lists. When given a large list to sort, the list is initially divided into P lists, and each processor sorts one of these lists. The lists are then exactly partitioned so that each of the elements in the new consecutive partitioned lists have values no smaller than any of the elements in the lists before it, nor larger than any of the elements in the list following it. Partitioning is done by P-1 processors. Each of the processors successively considers selected rows of elements from the sorted lists, and moves a partition boundary based on an element magnitude requirement and a partition size requirement. The new partitioned lists are then merged by the P processors, and simply strung together to provide a sorted list of all the elements.

EX1005, Abstract; *see also id.*, 10:66-11:33, FIGS. 3A, 3B.

Although Iyer does not expressly state that the initial list is a *file*, as claimed, based on the structure and handling of the data, Iyer describes the input list as a file both in form and function. EX1003, ¶ 65. For example, Iyer describes the lists as "data" accessed from DASD (Direct Access Storage Devices). *See* EX1005, 4:9-13 ("Data, comprising lists to be sorted, is stored in various

direct access storage devices, DASD 26 and 28, also coupled to bus 18. Each processor will request data from a DASD to be transferred to main storage 20."). Thus, the "list" originates from non-volatile storage, a hallmark of file-based input. Additionally, regarding the structure of the list, Iyer describes "sorted lists" of "N elements" that are read in rows and indexed—typical of file structure. *See, e.g.*, 1:6-3:51, 4:43-58. Further suggesting a *file*, Iyer teaches that its technique is designed for "lists of almost any size." EX1005, 10:44-51. In computer science, such scalable sorting tasks assume that input resides in external files, particularly when the input data size exceeds physical memory. EX1003, ¶ 65.

Accordingly, Iyer discloses, suggests, or renders obvious element [1.pre].

b. [1.1]: *"(a) automatically determining file allocation and logically subdividing records of said input file into a plurality of partitions;"*

Iyer discloses, suggests, or renders obvious this element. EX1003, ¶¶ 67-71. Specifically, Iyer discloses *automatically determining file allocation* (processor 1's receiving the initial list to be sorted) *and logically subdividing* (dividing the list into P partitions) *records* (N elements, such as addresses, in the initial list) *of said input file into a plurality of partitions* (P partitions):

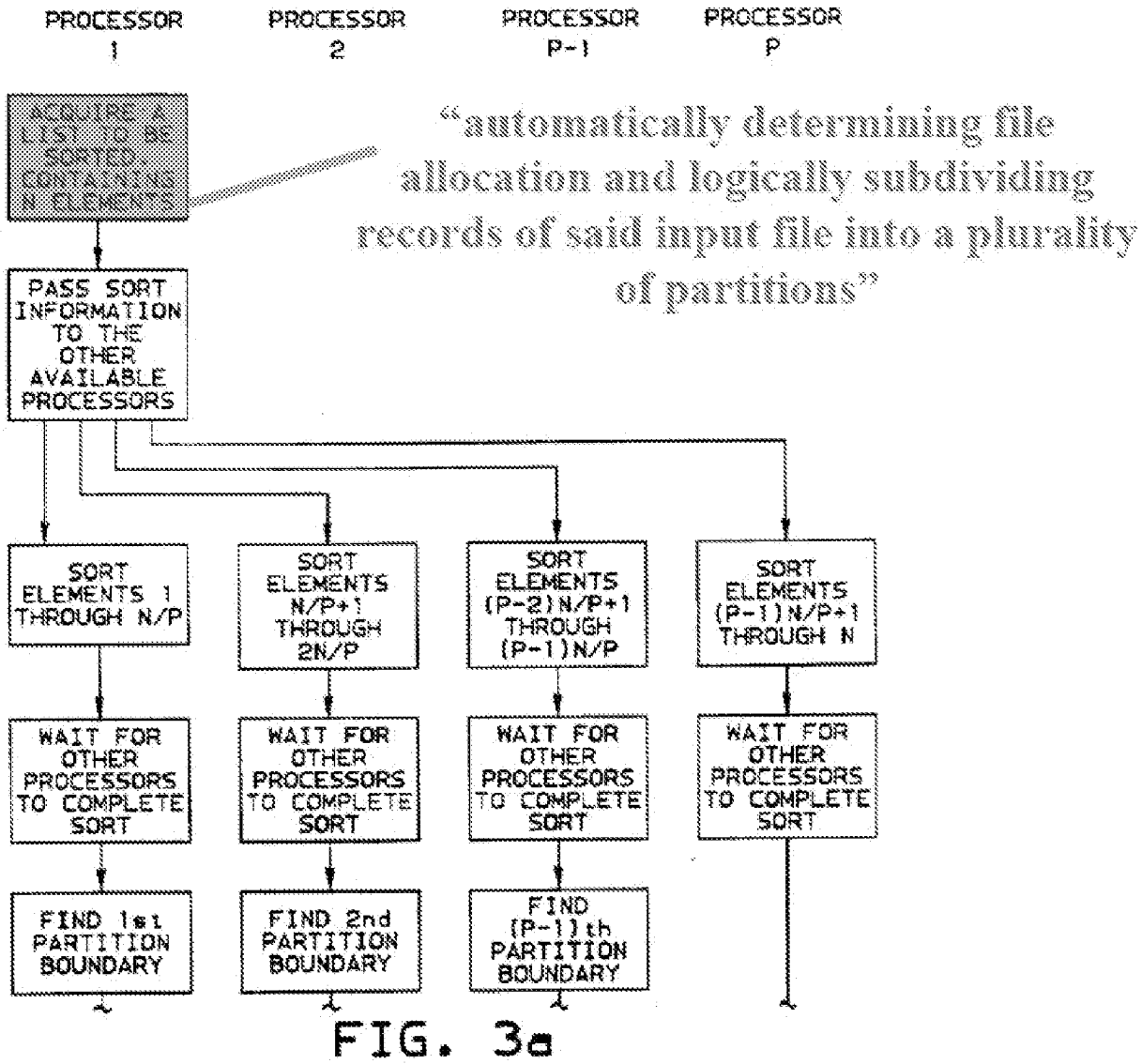


FIG. 3a

Iyer, FIG. 3A*

***Automatically determining file allocation and logically subdividing ... into a plurality of partitions*:** Iyer discloses this aspect by receiving the input list and dividing it into equal partitions of N elements. EX1003, ¶ 68. The '582 Patent does not explain what is meant by determining file "allocation". Thus, in my opinion, a POSITA would have reasonably broadly interpreted it to mean examining the data/file in some way to identify the elements/records to be distributed for parallel processing. *Id.* In this manner, Iyer teaches "dividing the work as equally as possible between the multiple processors" and that " [t]he first step is to divide the list into multiple equal sized lists." EX1005, 4:23-26. Iyer shows this process as the first step in Figure 3A:

A block diagram of the method of sorting a list using multiple processors is shown

> in FIG. 3A and 3B when placed end to end. Multiple processors 1, 2 . . . P-1, P are shown at the top of the figure, and the actions taken are directly in line beneath each of the processors. One of the processors, processor 1 acquires a list to be sorted containing N elements. It passes the sort information to the other processors, identifying which elements of the list are to be sorted by each of the processors. The elements to sort may also be determined by each processor if each processor has been preprogrammed to search predetermined portions given the size of a list.

Id., 10:66-11: 9.

> Any number of sorted lists are efficiently partitioned into P lists, where P represents the number of processors available to sort the resulting lists. When given a large list to sort, the list is initially divided into P lists, and each processor sorts one of these lists. When the lists are then partitioned in accordance with the present invention, each of the elements in the new lists created by partitioning have values no smaller than any of the elements in the partitions before it, nor larger than any of the elements in the partitions following it.

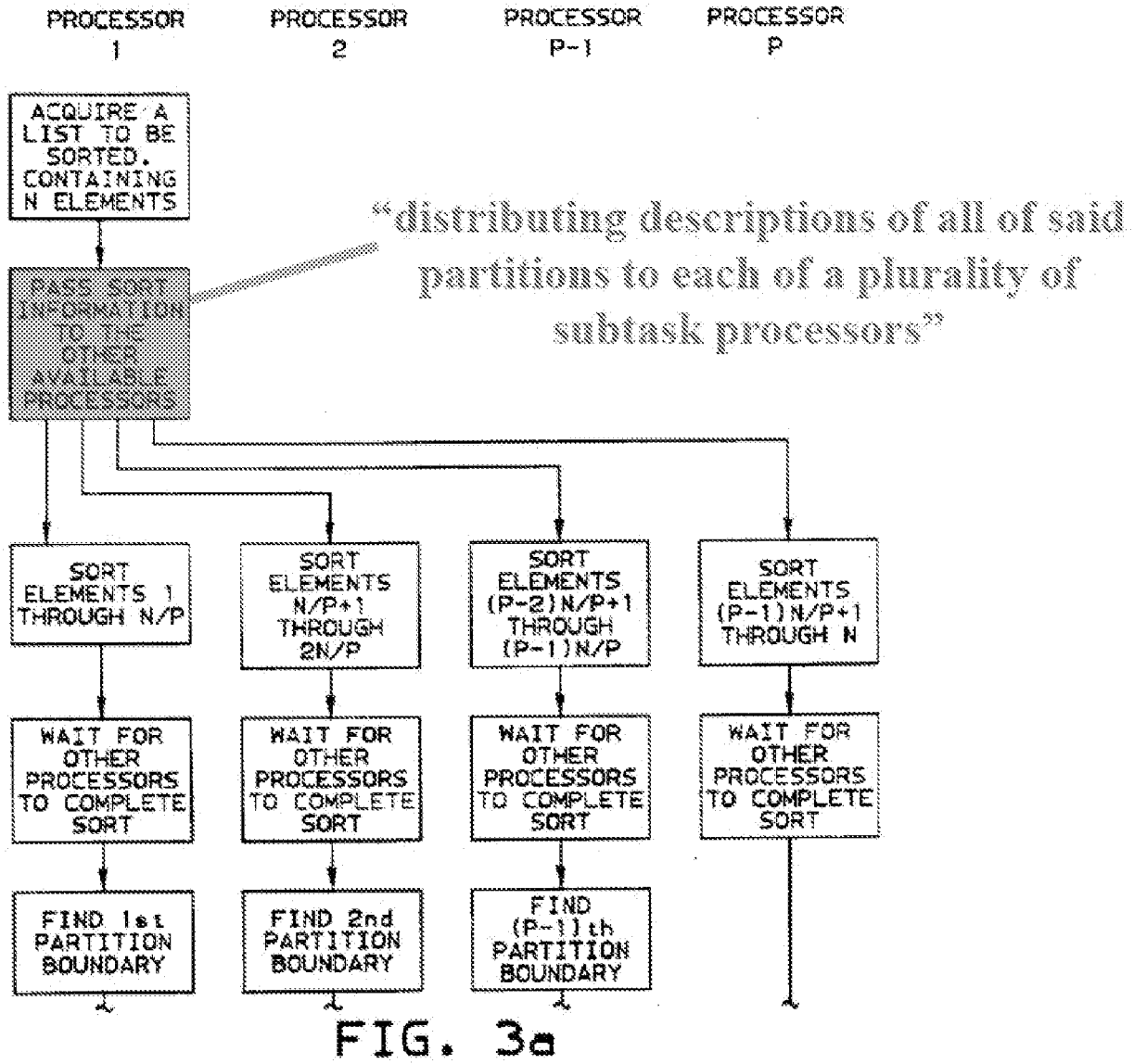
Id., 2:20-30; *see also id.*, Abstract. Although Iyer does not expressly state that the process is performed *automatically*, a POSITA would have understood it to be so because Iyer describes a computer-based merge/sort operation and nowhere describes any need for human input or intervention to complete the process. EX1003, ¶ 69.

***Records of said input file*:** Iyer teaches that the input list consists of N elements, thereby disclosing the claimed *records of said input file*. *Id.*, 11:2-4 ("One of the processors, processor 1 acquires a list to be sorted containing N elements."); *see also id.*, 1:10-34, 2:20-3:50, 4:36-5:10 (describing the list as containing "elements" to be sorted); EX1003, ¶ 70. Other than indicating that records are elements of the input file that are distributed to, and processed by, the parallel Sub-Task processors, the '582 Patent provides little detail around the meaning of the term record. *See, e.g.*, EX1001, 5:27-6:40. Thus, Iyer's list elements read on the claimed *records*. EX1003, ¶ 70.

Accordingly, Iyer discloses, suggests, or renders obvious element [1.1].

c. [1.2]: *"(b) distributing descriptions of all of said partitions to each of a plurality of subtask processors*;"

Iyer discloses, suggests, or renders obvious this element. EX1003, ¶¶ 72-74. Specifically, Iyer discloses *distributing descriptions* (passing sort information/identifications) *of all of said partitions* (of partitions 1-P*) to each of a plurality of subtask processors* (processors 1-P), as claimed.



Iyer, FIG. 3A*

Specifically, Iyer explains:

> **It passes the sort information to the other processors, identifying which elements of the list are to be sorted by each of the processors**. The elements to sort may also be determined by each processor if each processor has been preprogrammed to search predetermined portions given the size of a list.

EX1005, 11:2-6.

> The first step is to divide the list into multiple equal sized lists. If the items are represented by addresses, one processor usually provides a list of addresses to each of the other processors. The number of such lists should be equal to the number of processors available. **Each processor then takes a list and ....**

Id., 4:25-31.

Iyer's processor 1 passes descriptions of *all* partitions to *each* of the processors 1-P, as claimed. EX1003, ¶ 74. As Iyer explains, processor 1 "passes the sort information to the other processors, identifying which elements of the list are to be sorted by each of the processors." EX1005, 11:2–6. In other words, each processor receives the same "sort information" identifying the partition assigned to each processor—including itself. EX1003, ¶ 74. Accordingly, Iyer discloses, suggests, or renders obvious *distributing descriptions of* ***all*** *of said partitions to* ***each*** processor.

Accordingly, Iyer discloses, suggests, or renders obvious element [1.2].

d. [1.3]: "*c) simultaneously executing at least a respective one of the subtasks of the computer-executable process in each of at least some of said processors on a respective one of the partitions with each subtask reading and processing the respective partition so as to process the respective partition and produce respective subtask output and;*"

Iyer discloses, suggests, or renders obvious this element. EX1003, ¶¶ 76-78. Specifically, Iyer discloses *simultaneously executing at least a respective one of the subtasks* (sorting its partition of the list) *of the computer-executable process* (sorting process) *in each of at least some of said processors* (each of processors 1-P) *on a respective one of the partitions* (each processor's partition 1-P) *with each subtask* (processor 1-P) *reading and processing the respective partition* (partition 1-P) *so as to process the respective partition* (partition 1-P) *and produce respective subtask output* (sorted partition), as claimed:

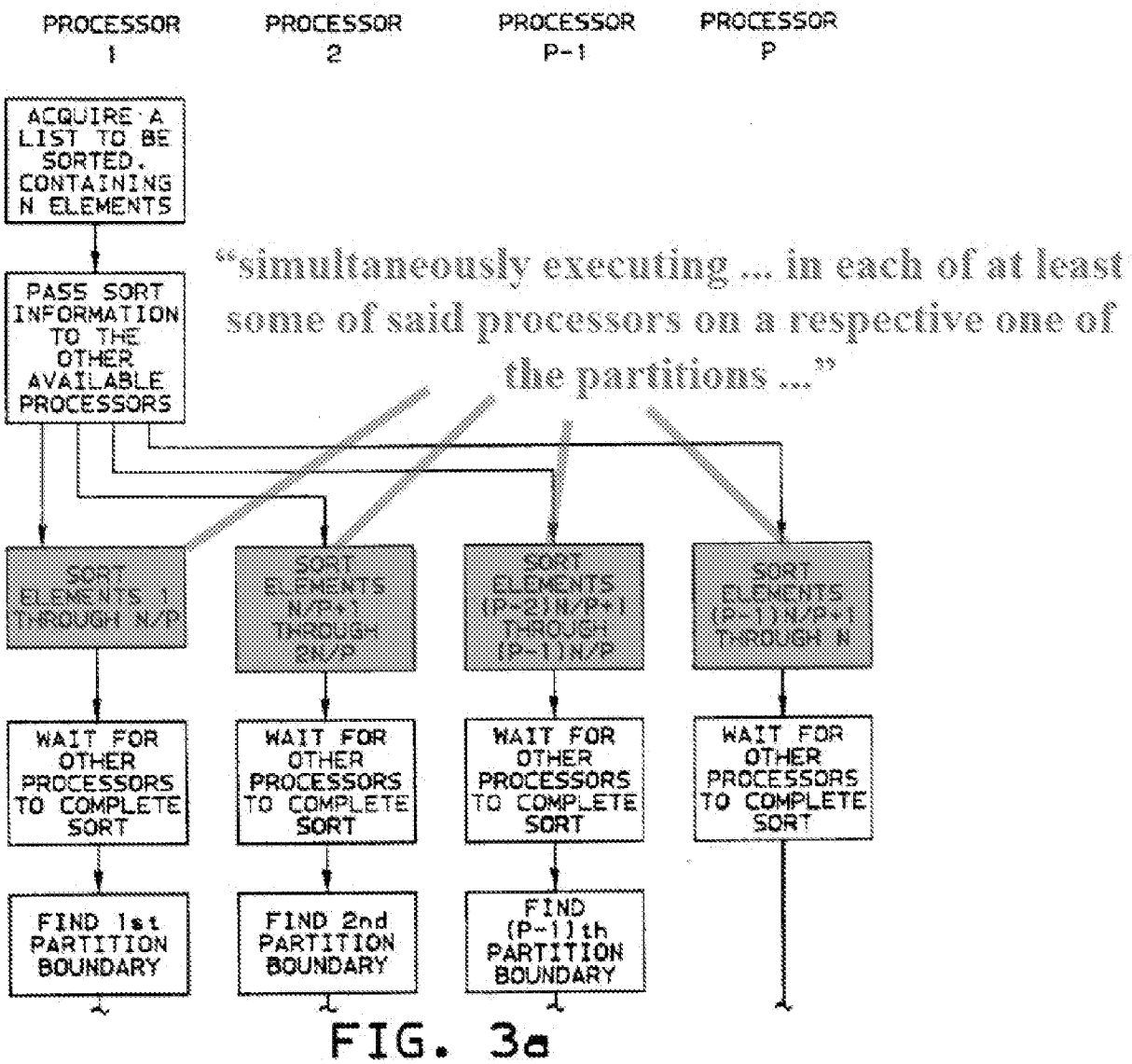


FIG. 3a

Iyer, FIG. 3A*

Regarding the simultaneous sorting, Iyer discloses:

> **Each processor will then sort a portion of the list**, and wait for the other processor to complete their sorts. The lists are sorted using a traditional sorting method such as quicksort, tournament tree, etc. Following the sort, P-1 of the processors will determine partition boundaries, and pass the partition information to the processor which is determining the next partition boundary. During the partitioning, processors will also need to perform sorts to determine which elements to move about the boundaries during the iterative process. Similar sorting methods as described above may be used. The amount of sorting is minimal since the lists are already individually sorted.

EX1005, 12:10-22.

> When given a large list to sort, the list is initially divided into P lists, **and each processor sorts one of these lists.** When the lists are then partitioned in accordance with the present invention, each of the elements in the new lists created by partitioning have values no smaller than any of the elements in the partitions before it, nor larger than any of the elements in the partitions following it.

Id., 2:22-30.

> **Each processor then takes a list and performs a normal sort of the list so that all the items in the list are in a desired order**. One such order is an ascending order based on the value of the digital representation of the items. Many such sort routines are commonly practiced today, and will not be discussed further.

Id., 4:29-35.

Accordingly, Iyer discloses, suggests, or renders obvious element [1.3].

e. [1.4] "*d) thereafter repeating step (c) in at least some of the subtask processors each with another unprocessed partition on a first-come/first-served basis; and*"

To the extent Iyer does not disclose this element, Reeve does through its multiprocessor GRAB strategy. It would have been obvious to modify Iyer's processors 1-P to use Reeve's GRAB strategy to sort and merge the partitions with a reasonable expectation of success, resulting in element [1.4]. EX1003, ¶¶ 79-95.

i. Reeve discloses element [1.4]

Reeve's GRAB strategy th*ereafter repeats step (c) in at least some of the subtask processors* (processor threads) *each with another unprocessed partition* (remaining unprocessed tile*) on a first-come/first-served basis* (via a first-come, first-serve basis). Specifically, Reeve discloses a load-balancing "GRAB strategy" in which distributed simultaneously "grab," lock, and execute sets of one or more tiles until all the tiles of the program have been executed. EX1003, ¶ 80. This is a passive load-sharing strategy, with no predetermined assignment of tasks to processors, because the processors grab the tiles for execution on "a first-come, first served basis." Regarding the characteristics of the GRAB strategy, Reeve explains:

> A grab strategy too divides the iteration space such that there are more tiles than available processors. The resultant tiles are assigned to requesting processors on a first-come-first-serve basis.

EX1006, 4:1-7.

> When using the GRAB strategy, Runtime Library 66 lets the scheduler handle all thread-processor bindings, load balancing, affinity, etc.

Id., 22:56-58.

GRAB strategy

> This strategy issues tiles on a first-come, first-serve basis to the thread group. Each thread must obtain a common lock to access a tile for execution. This strategy is designed to load balance between the threads of the thread group.
>
> Note that this strategy does not consider data affinity considerations when assigning tiles to processors, and that it would not be a good strategy to choose in conjunction with affinity regions.

Id., 26:14-22.

> The GRAB tiling strategy, is dynamic and will keep each thread busy as much as possible, but will probably cause data migration to occur more frequently. In addition to affinity loss problems is the additional synchronization overhead from the locking required to access a new tile.

Id., 26:32-36.

6.1.2 Locking during the grab strategy (critical section)

> The grab strategy requires all member threads to update a marker data structure to get the next available tile. This marker data structure is held in a lock.

Id., 42:28-31.

Accordingly, Reeve discloses element [1.4].

ii. Rationale to combine Iyer and Reeve

A POSITA would have been motivated to combine Iyer and Reeve with a reasonable expectation of success, resulting in element [1.4]. EX1003, ¶ 82. In particular, a POSITA would have modified Iyer's processors 1-P to implement Reeve's GRAB strategy to improve efficiency of Iyer's sort method.

- **Teaching, suggestion, motivation**

Iyer and Reeve provide teaching, suggestion, and motivation for this combination. *KSR Int'l Co. v. Teleflex Inc.*, 127 S. Ct. 1727 (2007). For example, a POSITA would recognize that Iyer's process repeatedly leaves processors 1-P idle and waiting at multiple stages of its process. EX1003, ¶ 83. As Iyer explains, after receiving its partition, "[e]ach processor will then sort a portion of the list, and **wait for** the other processor to complete their sorts." EX1005, 11:10-11. Next, when determining boundaries, "[t]he processors will **wait for** the partition boundary information from the preceding processor" before proceeding to the merge step. *Id.*, 11:23-24.

Finally, during merging, "[a] designated processor, processor 1 in this case, will then **wait for** the other processors to complete the merge of their respective partitions ...." *Id.*, 11:27-30.

In identifying the source of this downtime, a POSITA would recognize that Iyer's strategy "divid[es] the work as equally as possible between the multiple processors" by "divid[ing] the list into multiple equal sized lists" assigned as partitions. EX1005, 4:23-26. Because Iyer's strategy assigns each processor one partition of equal size, a POSITA would have understood that the waiting results from the processors' having different computing capacities. *See id.*, 4:28-29 ("The number of such lists should be equal to the number of processors available."). Accordingly, a POSITA would have looked for ways to better balance the load among processors 1-P. EX1003, ¶ 84.

Turning to Reeve, a POSITA would note Reeve's "GRAB strategy," which "is designed to load balance between the threads of the thread group." EX1006, 26:14-22. The GRAB strategy "is dynamic and will keep each thread busy as much as possible," EX1006, 26:32-36, because it "issues tiles on a first-come, first-serve basis to the thread group." *Id.*, 26:14-22. Thus, a POSITA would have understood that Reeve's GRAB strategy could be used to balance the load and reduce wait times of Iyer's processors 1-N. EX1003, ¶ 85.

Comparing the two references, a POSITA would observe that while Iyer's strategy divides the initial list into the same number of (equal) partitions as the number of processors 1-P, Reeve's "grab strategy ... divides the iteration space such that there are **more tiles than available processors**." EX1006, 4:1-7. From this, a POSITA would have understood that Iyer's processors would stay busy and wait less if the initial list were divided into a greater number of smaller (i.e., "bite-size") partitions—more than the number of processors 1-P. In this manner, as the processors complete processing of a current partition, the processors could dynamically grab a new partition to process on a "first-come, first-serve" basis, as in Reeve's GRAB strategy. In this way, a faster processor would process more partitions such that all processors reach the next stage at the same time. EX1003, ¶ 86.

Indeed, the advantages of self-scheduling—or first-come, first-served (FCFS)—techniques for multiprocessor systems were well known by at least the late 1980s, as described by Tang (EX1026). *Id.*, ¶ 87. For example:

Dynamic load balancing. Tang explains that "[o]ne of the advantages of dynamic scheduling is that it tends to balance the workload among processors, especially when the

execution time of each task varies substantially due to conditional statements in tasks or different execution rate of processors." EX1026, 2. This would have motivated a POSITA to apply Reeve's FCFS/self-scheduling approach to Iyer's system, particularly because Iyer's processors exhibit varying task execution times that result in multiple stages of idle time throughout the process. EX1003, ¶ 88.

Reduced compile-time information requirements. Tang further explains that "[d]ynamic scheduling does not need such [upfront compile-time] information. It assigns tasks to processors dynamically according to the available resources at run time." EX1026, 2. This contrasts with static scheduling, which requires extensive upfront knowledge—such as exact execution times or other parameters—that are often unavailable at compile time. *See, e.g.*, EX1026, 1–2, 6; EX1003, ¶ 89.

Implementation efficiency. Tang notes that "[s]elf-scheduling is an efficient dynamic scheduling scheme. In self-scheduling, processors schedule themselves at run time by accessing and updating some shared variables or tables without calling the operating system." EX1026, 2; *see also id.*, 10 ("Processor self-scheduling is an efficient dynamic scheduling for multiprocessors."); *see also* EX1003, ¶ 90.

Successful industry adoption: Tang also highlights real-world adoption, explaining that "[p]rocessor self-scheduling for singly-nested parallel loops has been implemented in several experimental multiprocessor optimizing compilers .... Alliant FX/8 and IBM 3081 ... also use self-scheduling for processor allocation." EX1026, 1; *see also* EX1003, ¶ 91.

These and other well-established benefits of self-scheduling/FCFS multiprocessor techniques would have motivated a POSITA to apply Reeve's GRAB strategy to implement Iyer's sorting method, with a reasonable expectation of success. EX1003, ¶ 92. To make the combination, a POSITA would have modified Iyer's processor 1 to break the initial list into more partitions than there are processors 1-P. *Id.* The POSITA further would have equipped each of processors 1-P to perform Reeve's GRAB strategy during at least the sort step shown in Figure 3A and discussed above. *Id.* The POSITA would have incorporated Reeve's mechanism of "[l]ocking during the grab strategy" to have Iyer's processors "update a marker data structure to get the next available

[partition]" upon finishing sorting a current partition. EX1006, 42:29-31; *see also* EX1003, ¶ 92[5]. As a result, each processor 1-P would "grab" a new partition upon completing the sort of its current partition, rather than sorting a larger partition and waiting for other processors to complete their work. EX1003, ¶ 92. Additionally, the combination would also parallelize the merge steps by creating on-demand merge tasks formed by combining outputs from prior sort steps. *Id.*

A POSITA could have done this with a reasonable expectation of success because both Iyer and Reeve are directed to multiprocessor systems designed to balance load among many processors. *Id.* Further, Reeve's GRAB strategy is directly applicable to Iyer's sorting process because sorting and merging records do not involve "data affinity" or interdependencies—each record can be processed independently without regard to the others. *See, e.g.* EX1006, 26:14-22, 26:32-36 (describing the GRAB process as appropriate for such applications); EX1003, ¶ 93.

- **Use of Known Technique to Improve Similar Devices (Methods, or Products) in the Same Way**

Further, Requester's proposed combination of Iyer and Reeve simply applies a known technique (Reeve's GRAB strategy in a multiprocessor system) to improve a similar system (Iyer's multiprocessor architecture) in the same way (keeping processors busy by taking work on a "first-come, first-serve" basis). *KSR*, 550 U.S. 398, 417 (2007); MPEP § 2143.C (identifying factors for the "known technique" rationale); EX1003, ¶ 94. As discussed, Iyer describes a "base" multiprocessor system that divides a to-be-sorted list into equal partitions matching the number of available processors, which requires each processor to wait for the other processors to complete their work before moving to the next step. Reeve discloses a comparable multiprocessor system but improves on Iyer's approach by breaking the workload into smaller partitions—more than the number of processors. This allows each processor to "grab" a new partition upon completing its current one—on a first-come, first-serve basis—so that all processors remain busy until the job is complete. EX1003, ¶ 94. A POSITA could have predictably applied Reeve's GRAB technique to at least Iyer's initial partitioning and sorting process, resulting in less processor idle time and greater processing efficiency.

---

[5] For this additional reason, the Iyer-Reeve combination would further satisfy element [1.2] because all processors 1-P would have access to the sort information (partition *descriptions*) via the marker data structure of Reeve, incorporated into Iyer's system. EX1003, ¶ 92.

Accordingly, Iyer and Reeve render obvious element [1.4].

f. [1.5] *"(e) generating at least one output combining all of the subtask outputs and reflecting the processing of all of said subtasks."*

Iyer discloses, suggests, or renders obvious this element. EX1003, ¶¶ 96-99. Specifically, Iyer discloses *generating at least one output* (finding sorted partition boundaries, merging, and concatenating each of the sorted P partitions) *combining all of the subtask outputs* (sorted P partitions) *and reflecting the processing of all of said subtasks* (processors 1-P), as claimed:

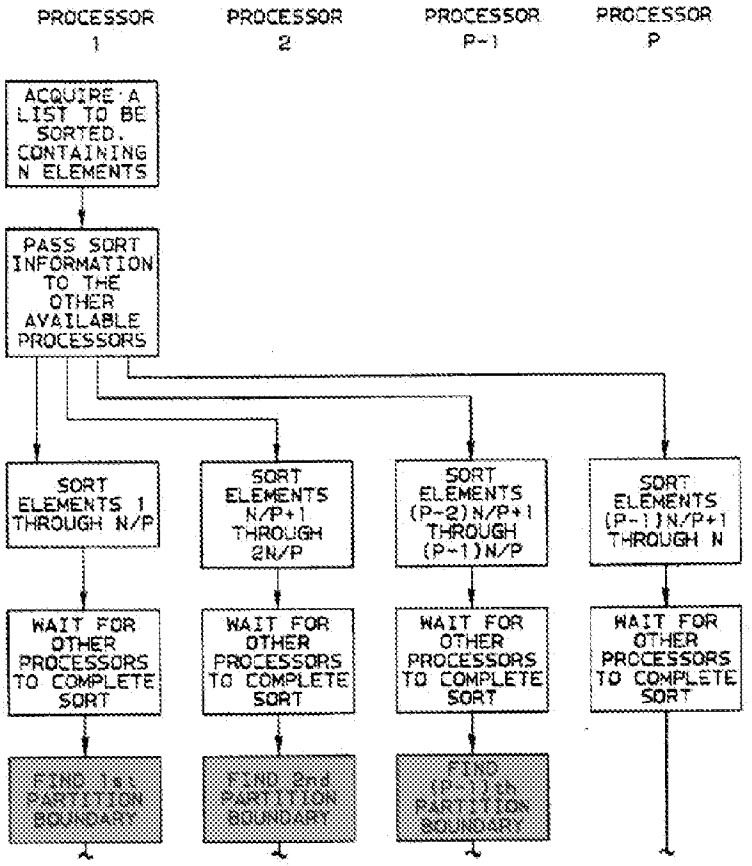


FIG. 3a

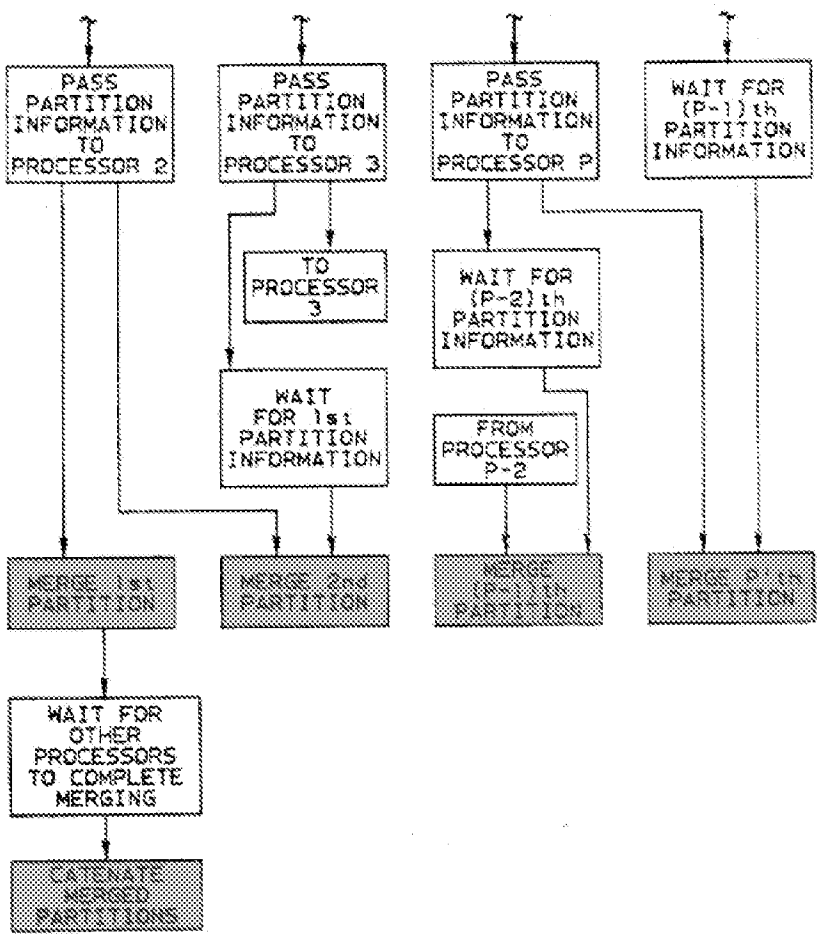


FIG. 3b

Iyer, FIGS. 3A, 3B*

Specifically, Iyer discloses:

> **Following the sort, P-1 of the processors will determine partition boundaries**, and pass the partition information to the processor which is determining the next partition boundary. During the partitioning, processors will also need to perform sorts to determine which elements to move about the boundaries during the iterative process. Similar sorting methods as described above may be used. The amount of sorting is minimal since the lists are already individually sorted.
>
> The processors will wait for the partition boundary information from the preceding processor. **When both boundaries are identified, each of the processors will then merge one of the partitions which is defined by the boundaries**. **A designated processor, processor 1 in this case, will then wait for the other processors to complete the merge of their respective partitions, and then catenate the merged partitions. The result is the actual list. This result may be simply a list of addresses of the elements in the correct order, or the elements may actually be rearranged in storage to be in the correct order.**

EX1005, 11:13-33.

Regarding the merging and concatenating process, Iyer further discloses:

The lists in each partition are then merged by the P processors, and simply strung together to provide a sorted list of all the elements.

The first partition consists of all the elements of all the lists which are below a certain value which results in the partition containing N/P elements. The second partition consists of all the elements at or above the first value, but below a second higher value which results again in the partition containing N/P elements. This continues until the Pth partition contains the highest value, and again, N/P elements. The lists in a partition are merged by a single processor, each of P processors being assigned the task of merging the lists in one partition. When the merge processing is complete, the merged lists are simply strung together to produce the final sorted list. As can be seen, the time intensive merging operation is performed by all available processors simultaneously, rather than in several partition and merge operations as in the prior art. Also each partition is nearly the same size so no processor load imbalance occurs as in the prior art.

To determine the partition boundaries in the preferred embodiment, we begin with a set of null lists. A calculation is performed to find the approximate middle element of each list and this row is added for consideration with the initial partition prior to the single element in each list. The maximum element of those elements immediately prior to the partition in each list is determined. The number of elements immediately following the partition which are less than this maximum element is also determined. This value is the number of elements which must be moved from the bottom partition into the top partition to assure that all the elements of the top partition are less than all the elements of the bottom partition. This criteria is hereafter referred to as the magnitude requirement.

Next, the number of elements that must be moved from the bottom partition into the top partition in order to assure that the size of the partitions are in the correct proportion is determined. This criteria is hereafter referred to as the partition-size requirement. These two values, the number of elements that must be moved to guarantee the magnitude requirement, and the number of elements that must be moved to guarantee the partition-size requirement, are then compared. The comparison will determine whether elements need be moved from the top partition into the bottom partition or vice-versa in order to guarantee both the magnitude and partition-size requirements. If elements need be moved from the top partition into the bottom then the partition is adjusted such that the appropriate number of largest elements from the top partition are moved into the bottom. If elements need be moved from the bottom partition into the top then the partition is adjusted such that the appropriate number of smallest elements from the bottom partition are moved into the top. The partition is now correct for the elements that are currently being considered in the lists.

In the next step, two more rows, each halfway between the first and last rows, and the middle row are added for consideration, and the maximum of those elements

> immediately prior to the partition boundary is again determined. The process continues as described above, and the result is a correct partition for the elements that are currently being considered in the lists. During each of the succeeding steps, new rows are added for consideration, until finally, after log2 n steps (where n is the number of elements in the longest list to be partitioned), all the rows have been added. After this final iteration, the partition is finally determined.
>
> p-1 processors perform this operation in parallel to find P-1 partition boundaries, and hence define P partitions each containing one or more lists for merging. Each of the P partitions may contain elements from one or more of the original sorted lists. The operation is the same in each of the processors, but a weighing factor is applied to the size requirement, and each processor ends up moving different elements in each of the steps, so that the resulting partition boundaries calculated are different.
>
> Since the number of elements in each resulting partition differs from the others by at most 1 element, work is divided evenly between the processors for the final merge phase. In the final merge phase, each of the P processors merges the lists in its partition to form P sorted lists. The lists are then simply strung together to form the final sorted lists. Maximum use of processors is obtained, and hence a near linear improvement in sort time is obtained when adding further processors.

EX1005, 2:29-3:42.

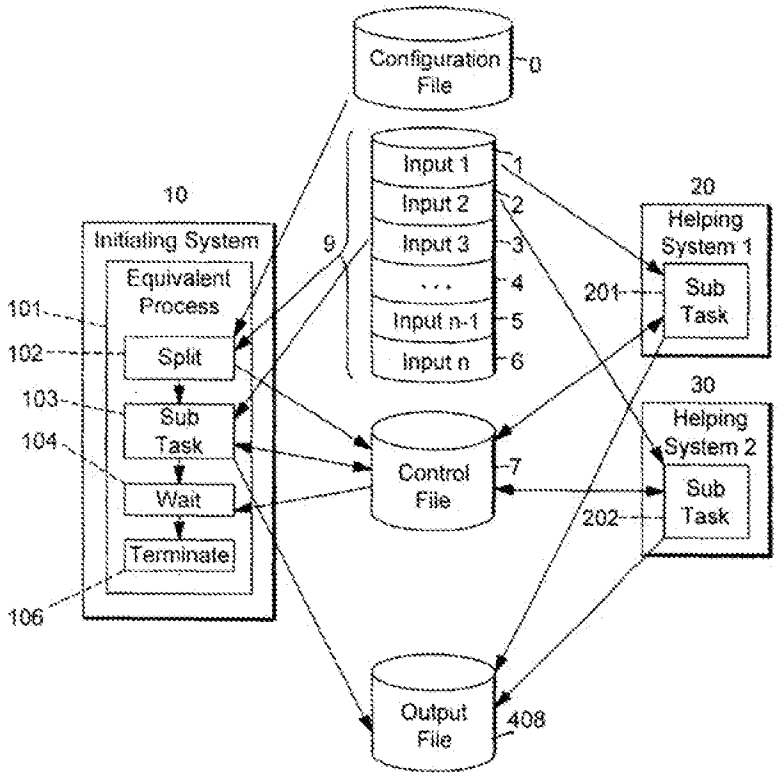
Accordingly, Iyer discloses, suggests, or renders obvious element [1.5].

2. Dependent Claim 2

a. [2.1]: "*The method defined in claim 1 wherein the automatic determination of file allocation and logical subdivision of records of said input file into said plurality of partitions in step (a) and the distribution of the description of all of said partitions in step (b) is carried out with at least one further processor in addition to the subtask processors.*

As explained above, Iyer and Reeve render claim 1 unpatentable. *Supra* Section VI.A.1. Moreover, Iyer discloses, suggests, or renders obvious the additional elements of dependent claim 2. EX1003, ¶¶ 100-102. Specifically, Iyer discloses that *steps (a) and (b)* of claim 1 are carried out with at least one further processor (processor 1) in addition to the subtask processors (processors 2-P), as claimed. As explained above for element [1.4], Iyer's processors 2-P correspond to the claimed *subtask processors*. Moreover, Iyer's first processor, processor 1, corresponds to the claimed *at least one further processor* recited in claim 2. As Iyer explains, processor 1 not only partitions the initial list and passes the partition information to the remaining processors 2-P, but it also sorts, merges, and concatenate its own partition of the initial list. *See, e.g.*, EX1005, 10:66-11:40; FIGS. 3A-3B; EX1003, ¶ 100.

Iyer's configuration is similar to that of the '582 Patent which has an initiating system 10 (*at least one further processor*) that creates the partitions and multiple helping systems 20, 30 (*subtask processors*) that perform the tasks (*see* EX1003, ¶ 101):



'582 Patent, FIG. 3

Accordingly, Iyer and Reeve render obvious claim 2. EX1003, ¶ 102.

3. Dependent Claim 3

a. [3.1]: "*The method defined in claim 1, further comprising the step of merging said subtask outputs in step (e).*"

As explained above, Iyer and Reeve render claim 1 unpatentable. *Supra* Section VI.A.I. Moreover, Iyer discloses, suggests, or renders obvious the additional elements of dependent claim 3. EX1003, ¶¶ 103-105. Specifically, Iyer discloses *the step of merging* (merging and concatenating) *said subtask outputs* (sorted partitions) *in step (e).*

As explained above for element 1.5, Iyer discloses the claimed *step (e)* by one or more of each processor finding its sorted partition boundaries, merging its partition, and concatenating its

petition to create the final list. Thus, Iyer's step (e) includes *merging* its output, as claimed. In particular, at least each processor's merging and concatenating discloses the claimed *merging*:

> Following the sort, P-1 of the processors will determine partition boundaries, and pass the partition information to the processor which is determining the next partition boundary. During the partitioning, processors will also need to perform sorts to determine which elements to move about the boundaries during the iterative process. Similar sorting methods as described above may be used. The amount of sorting is minimal since the lists are already individually sorted.
>
> The processors will wait for the partition boundary information from the preceding processor. **When both boundaries are identified, each of the processors will then merge one of the partitions which is defined by the boundaries**. **A designated processor, processor 1 in this case, will then wait for the other processors to complete the merge of their respective partitions, and then catenate the merged partitions. The result is the actual list. This result may be simply a list of addresses of the elements in the correct order, or the elements may actually be rearranged in storage to be in the correct order.**

EX1005, 11:13-33; *see also* EX1003, ¶ 104.

Accordingly, Iyer and Reeve render obvious claim 3.

4. Dependent Claim 4

[4.1] "*The method defined in claim 1 wherein the output in step (e) is a succession of outputs from said subtasks in a one-to-one correspondence with said records of said input file.*"

As explained above, Iyer and Reeve render claim 1 unpatentable. *Supra* Section VI.A.I. Moreover, Iyer discloses, suggests, or renders obvious the additional elements of dependent claim 4. EX1003, ¶¶ 106-107. Specifically, Iyer discloses that *the output in step (e)* (sorted, merged, and concatenated final list) *is a succession of outputs from said subtasks* (sorted individual partitions) *in a one-to-one correspondence with said records* (elements) *of said input file* (initial list). Iyer discloses that the final list is simply the original list with its elements in the "correct order":

> The result is the actual list. This result may be simply a list of addresses of the elements in the correct order, or the elements may actually be rearranged in storage to be in the correct order.

EX1005, 11:30-33. In other words, "[w]hen the merge processing is complete, the merged lists are simply strung together to produce the final sorted list." *Id.*, 2:44-46. Because the final list contains all the elements of the initial list—just reordered—there is a *one-to-one correspondence* between them. EX1003, ¶ 106. That is, each element in the initial list appears once in the final list, and vice versa, satisfying the claimed *one-to-one correspondence*, as claimed. *Id.* This aligns with the '582

Patent, which also describes a *one-to-one correspondence* between the input and output lists when all records from the input file appear in the output file. *See, e.g.*, EX1001, 2:21-25, 5:27-38.

Accordingly, Iyer and Reeve render obvious claim 4.

5. Dependent Claim 5

[5.1] "*The method defined in claim 1 wherein the output in step (e) is an accumulation of output records from said subtasks in an arbitrary order.*"

As explained above, Iyer and Reeve render claim 1 unpatentable. *Supra* Section VI.A.I. Moreover, Iyer discloses, suggests, or renders obvious the additional elements of dependent claim 5. EX1003, ¶¶ 108-110. Specifically, Iyer discloses that *the output in step (e)* (sorted, merged, and concatenated final list) *is an accumulation of output records* (elements in each sorted partition P) *from said subtasks* (processors 2-P) *in an arbitrary order* (finished at different times). EX1003, ¶ 108.

Iyer discloses that "[e]ach processor will then sort a portion of the list, and **wait for** the other processor **to complete** their sorts." EX1005, 11:10-11 Further, "[t]he processors will **wait for** the partition boundary information from the preceding processor." *Id.*, 11:23-24. Finally, "[a] designated processor, processor 1 in this case, will then **wait for** the other processors **to complete** the merge of their respective partitions." *Id.*, 11:27-30. This repeating "waiting" throughout the process confirms that Iyer's processors 1–P complete their respective sorting, partitioning, and merging subtasks independently and at varying times—i.e., in arbitrary order. Thus, the final output list is an *accumulation of output records* (elements) *from said subtasks* (processors 2-P) *in an arbitrary order*, as claimed. EX1003, ¶ 109.

Accordingly, Iyer and Reeve render obvious claim 5.

6. Dependent Claims 6 and 7

[6.1] "*The method defined in claim 1 wherein said input file resides on a storage area network and is derived therefrom.*"

[7.1] "*The method defined in claim 1 wherein said input file resides on a network-attached storage and is derived therefrom.*"

As explained above, Iyer and Reeve render claim 1 unpatentable. *Supra* Section VI.A.I. Moreover, Iyer discloses, suggests, or renders obvious the additional elements of dependent claims 6 and 7. EX1003, ¶¶ 11-114. Regarding claim 6, Iyer discloses that *said input file* (initial list)

resides on a storage area network (DASD 26, 28) connected over I/O bus network 18) *and is derived therefrom*.

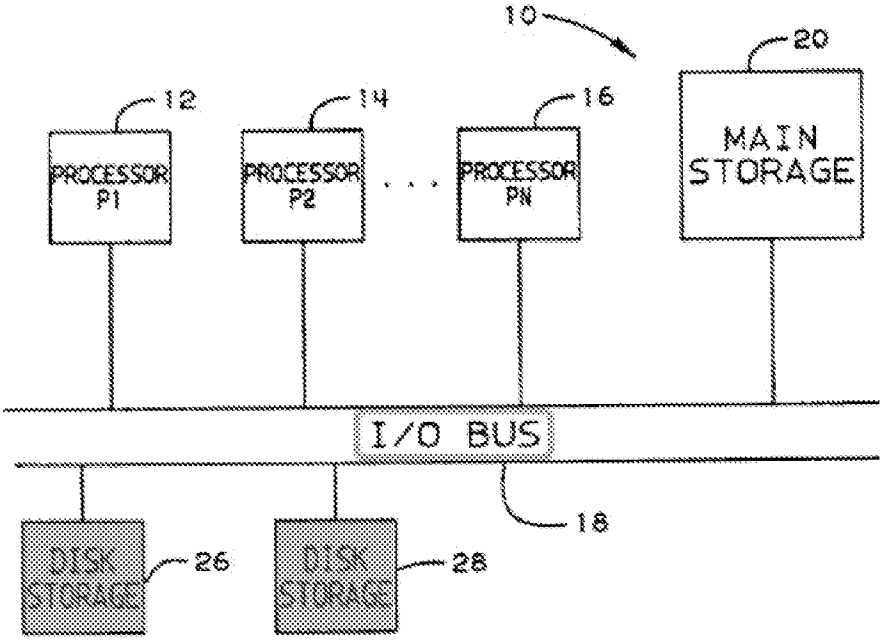


FIG. 1

Iyer, FIG. 1*

Iyer explains that the input file (list to be sorted) is derived from storage devices 26, 28 accessible over an I/O bus network 18:

> One computer system which can take advantage of the method of partitioning several sorted lists is shown generally at 10 in FIG. 1. A plurality of processors, P1, P2 . . . through PN numbered 12, 14, and 16, are shown **coupled to an I/O bus 18**. Each processor has access to a main storage 20, either by direct coupling, or through the I/O bus 18. **Data, comprising lists to be sorted, is stored in various direct access storage devices, DASD 26 and 28, also coupled to bus 18. Each processor will request data from a DASD to be transferred to main storage 20.**

EX1005, 4:7-13.

Iyer's DASD 26 and 28 disclose, suggest, or render obvious the *claimed storage area network* (SAN). EX1003, ¶ 113. These DASDs function as a centralized, shared storage system

accessed by multiple processors (1–N) via a common I/O bus 18, consistent with the defining features of a SAN. *See* EX1009; EX1003, ¶ 113. Each processor independently retrieves data from DASD 26 and 28 over bus network 18 to local memory, enabling parallel processing of a large dataset that is not stored locally—behavior consistent with SAN architecture, which decouples compute from storage via high-throughput interconnects. Although the term "SAN" was not in widespread use at the time Iyer was filed in 1989, Iyer shows that use of shared DASD in multiprocessor systems was well known. For example, Blelloch (EX1019) describes memory models and load balancing techniques for multiprocessors that use storage area networks and network-attached storage. *See* EX1019. As such, implementing Iyer's disclosed architecture using SAN principles would have been a predictable and obvious design choice for a person of ordinary skill in the art. EX1003, ¶ 113.

Regarding claim 7, Iyer's DASD 26 and 28 also disclose, suggest, or render obvious the claimed *network-attached* storage. EX1003, ¶ 114. Specifically, they serve as centralized repositories for data to be sorted, accessed by multiple processors over a shared I/O. NAS systems, like SANs, enable centralized storage to be accessed concurrently by multiple computing nodes, typically at the file level over a network. EX1009. Iyer describes its processors as accessing data from shared DASD 26 and 28 for local use—behavior consistent with applicable to file-level access provided by NAS devices. EX1003, ¶ 114. Furthermore, Iyer explicitly states the architecture is not dependent on any specific configuration and can include "loosely coupled processors," encompassing distributed systems with NAS-style access over a network. EX1005, 4:16-20. Thus, a POSITA would recognize that DASD 26, 28 are, or would have been, implemented as network-attached storage devices providing shared, file-level access to support the parallel sort-and-merge operations described by Iyer. EX1003, ¶ 114; *see also* EX1019 (using storage area networks and/or network-attached storage in load balancing techniques).

7. Dependent Claims 8-11

a. [8.1]: *"The method defined in claim 1 wherein said computer-executable process is a sort process."*

[9.1]: "*The method defined in claim 1 wherein said computer-executable process is a statistical analysis process."*

[10.1]: "*The method defined in claim 1 wherein said computer-executable process is a report-creating process."*

[11.1] "*The method defined in claim 1 wherein said computer-executable process includes a database query."*

During original prosecution, the Examiner rejected claims 8-11 as unpatentable over the Kitsuregawa reference because each merely recites an intended use of the claimed method without further meaningfully limiting the claimed method: use in a sort process, a statistical analysis process, a report-creating process, and a database query, respectively. *See* EX1002, 59. The Applicant did not dispute the Examiner's rejection or characterization of claims 8-11. Indeed, the Examiner was correct. EX1003, ¶¶ 115-118.

The preamble of independent claim 1 calls for a method of affecting "a computer-executable process" on a preexisting input file—i.e., an intended use of the claimed method on a "computer-executable process." Dependent claims 8-11 simply add further preamble labeling that process a "sort" process, a "statistical analysis" process, a "report-creating" process, and a "database query," respectively. But neither the body of independent claim 1 nor the bodies of dependent claims 8-11 recites any further process steps or structural elements related to such use in those processes—or any other type of computer-executable process. Because "the body of [the] claim fully and intrinsically sets forth all of the limitations of the claimed invention, and the preamble merely states ... the purpose or intended use of the invention, rather than any distinct definition of any of the claimed invention's limitations," "the preamble is not considered a limitation and is of no significance to claim construction." MPEP § 2111.02(II). Applicant's failure to rely on the intended uses recited in claims 8-11 to distinguish prior art confirms claims 8-10 are not entitled to patentable weight because "[w]ithout such reliance, ... a preamble generally is not limiting when the claim body describes a structurally complete invention such that deletion of the preamble phrase does not affect the structure or steps of the claimed invention." *Id.* (citing *Catalina Mktg. Int'l v. Coolsavings.com, Inc.*, 289 F.3d 801, 808 (Fed. Cir. 2002).

Indeed, as Dr. Weissman explains, the claimed technique could be used on many different types of computer-implemented processes besides the examples mentioned in claims 8-11. EX1003, ¶ 116. And it was well known in the art to use parallel processing techniques in each of the applications recited in claims 8-11:

- **Claim 8:** *See, e.g.*, EX1005, EX1011, EX1012, EX1015, EX1017, EX1022, and (disclosing different types of parallel sorting techniques).
- **Claim 9:** *See, e.g.,* EX1016 and EX1018 (disclosing parallel statistical analysis methods such as least median of squares and isotonic regression).
- **Claim 10:** *See, e.g.*, EX1022 (disclosing parallel report generation using distributed file systems).
- **Claim 11:** *See, e.g.*, EX1013 (disclosing parallel database query execution).

EX1003, ¶ 116.

Consistent with this, the specification of the '582 Patent teaches that the claimed method generally applies to "the field of sharing data and workload" among computer systems, highlighting that "[s]ort applications, statistical analysis batch applications and report writing applications and database queries are [mere] *examples* of applications that can readily enjoy this invention." Nothing in the '582 Patent claims or specification suggests that the claimed method is limited to the four claimed potential uses. Accordingly, for the reasons above, claims 8-11 are not entitled to patentable weight, and Iyer renders these claims unpatentable. Indeed, Iyer expressly relates to a *sort process*, and thus expressly discloses the subject matter of claim 8. EX1003, ¶ 117.

Accordingly, Iyer and Reeve render obvious claims 8-11.

8. Dependent Claim 12

a. [12.1]: "*The method defined in claim 2 wherein said one processor is part of a mainframe computer and the other processors are processors of at least one other computer.*"

As discussed, Iyer and Reeve render claim 2 unpatentable. *Supra* Section VI.A.2 In addition, Iyer discloses, suggests, or renders obvious the additional limitations of dependent claim 12. EX1003, ¶¶ 119-121. As explained above with respect to claim 2, Iyer's "processor 1" corresponds to the claimed "*said one processor*," and processors 2 through P correspond to the claimed "*other processors*." *See supra* Section VI.A.2.

Although Iyer does not specify the precise physical embodiments of processor 1 or processors 2–P, it clearly assigns processor 1 additional responsibilities beyond those of the remaining processors 2-P. EX1003, ¶ 120. Processor 1 not only performs the same sorting and merging tasks as processors 2–P but carries out additional tasks such as partitioning the initial list, distributing partitions, and compiling the final sorted list. *Id.* For example:

> **One of the processors, processor 1** acquires a list to be sorted containing N elements. It passes the sort information to the other processors, identifying which elements of the list are to be sorted by each of the processors.

EX1005, 11:2-6.

> **A designated processor, processor 1 in this case,** will then wait for the other processors to complete the merge of their respective partitions, and then catenate the merged partitions.

Id., 11:27-30.

Because processor 1 performs more functions and demands greater computing resources than processors 2–P, it would have been obvious to implement processor 1 on its own *mainframe computer*—on *at least one other computer* from that hosting processors 2-P—as recited in the claim. EX1003, ¶ 121; *see also* EX1008, 6 (a mainframe computer is a high-capacity, typically expensive system designed to handle intensive computational workloads). A POSITA would have understood that early parallel processing systems commonly employed a *mainframe computer* for the primary processor managing a multiprocessor task—one example being Reeve's dedicated compilation system 60—which is separate from the subtask processors. EX1003, ¶ 121. As other examples, EX1014 and EX1020 disclose parallel architectures in which a mainframe or master computer supervises or coordinates parallel processors of at least one other computer. *Id.* Thus, even though Iyer does not expressly describe processor 1 as implemented on a *mainframe computer*, doing so would have been a well-known and practical configuration at the time. *Id.* Given that processor 1 required some physical implementation, a POSITA would have been motivated—and found it obvious—to implement processor 1 using *a mainframe computer*. *Id.*

9. Dependent Claim 13

a. [13.1] "*The method defined in claim 1 wherein said plurality of processors are all parts of a single multiprocessor.*"

Iyer and Reeve render claim 1 unpatentable. *Supra* Section VI.A.I. Moreover, Iyer discloses, suggests, or renders obvious the additional element of dependent claim 13. EX1003, ¶¶ 122-123. Specifically, Iyer discloses that *said plurality of processors* (processors 1-P) *are all*

parts of a single multiprocessor, as claimed. EX1003, ¶ 122. Indeed, Iyer's title reads "Partitioning of sorted lists **for multiprocessors** sort and merge." And Iyer describes its Figure 1 system as the "**multiprocessor** computer system for performing the partitioning method of the present invention." EX1005, 3:55-57. As additional examples, EX1010, EX1011, EX1012, and EX1014 disclose parallel processors operating within a single multiprocessor system architecture. EX1003, ¶ 122.

10. Dependent Claim 14

a. [14.1] "*The method defined in claim 1 wherein the automatic determination of file allocation and logical subdivision of records of said input file into said plurality of partitions in step (a) and the distribution of the descriptions of all of said partitions in step (b) is carried out with at least one processor, and said one processor and said plurality of processors are all parts of a single multiprocessor not including said one processor.*"

Iyer and Reeve render claim 1 unpatentable. *Supra* Section VI.A.I. Iyer also discloses, suggests, or renders obvious the additional element of dependent claim 14. EX1003, ¶¶ 124-126. As an initial matter, claim 14 is inoperable and/or nonsensical because it simultaneously requires the one processor to *both* be part of *and* not be part of a single microprocessor, which is logically impossible. EX1003, ¶ 124.

Moreover, to the extent Applicant intended claim 14 to recite that the *single multiprocessor* that includes the *plurality of processors* does not also include *the one processor* that performs the claimed steps (a) and (b), Iyer discloses it. First, as explained for claim 13, Iyer describes its Figure 1 system as a "**multiprocessor** computer system." EX1005, 3:55-57. Second, as explained above for elements [1.1] and [1.2], processor 1 of Iyer performs the claimed steps (a) and (b). As explained above for claim 12, it would have been obvious to implement this processor 1 as a *mainframe computer* separate from the computer hosting processors 2-P given processor 1's additional responsibilities and higher resource requirements. *Supra* Section VI.A.8 This configuration also renders obvious claim 14, as the computer system hosting processors 2-P is distinct from—i.e., *not including*—the processor of the *mainframe computer*. EX1003, ¶ 125.

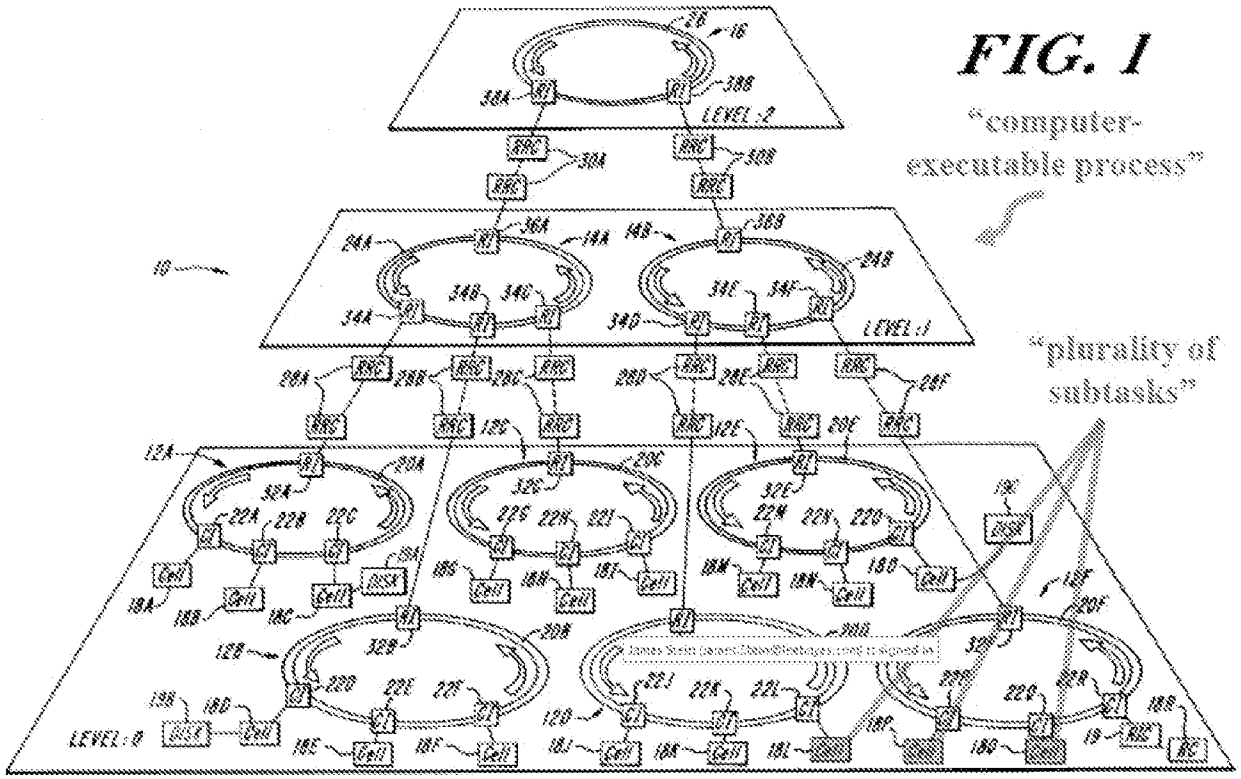
Accordingly, Iyer and Reeve render obvious claim 14.

B. Grounds 2 and 3: Reeve Anticipates and/or Renders Obvious Claims 1-14

1. Independent Claim 1

a. [1.pre]: "*A method of effecting on a preexisting input file a computer-executable process comprised of a plurality of subtasks, the method comprising the steps of:*"

To the extent the preamble is found to be limiting, Reeve discloses *a method of effecting on a preexisting input file* (Fortran program file) *a computer-executable process* (parallel processing of an input Fortran program) *comprised of a plurality of subtasks* (processors processing threads). EX1003, ¶¶ 127-131.



Reeve, FIG. 1*

***Computer-executable process*:** Reeve is directed to "an improved mechanism for executing programs on parallel processing computers, including those which are massively parallel." EX1006, 1:59-61. Reeve's process works by "executing an iterative sequence of instructions by arranging the sequence into subtasks and allocating those to the processors." *Id.*, 2:3-6.

Plurality of subtasks: In the '582 Patent, the plurality of "Sub Tasks" are the parallel processors executing portions of the distributed load. *See* EX1001, 6:14-40, FIG. 4 (describing

Sub Task 1 and Sub Task 2 running on processors and processing the partitioned load file 0 comprising Inputs 1-6). Similarly, as shown in Figure 1 of Reeve above, Reeve's system 10 distributes portions of the file to-be-processed—which Reeve calls "tiles"—to a plurality of "processing cells" 18A-F. EX1006, 5:23-52.[6] As Reeve's explains, "[e]ach of those cells include[s] a central processing unit and a memory element, interconnected along an intracellular processor bus (not shown)," and "the memory element contained in each cells stores all control and data signals used by its associated central processing unit." *Id.*, 5:36-43. During parallel execution of the program, these cells or processors form "thread groups" that execute portions of the file, which Reeve calls "tiles." *Id.*, 43:26-44.

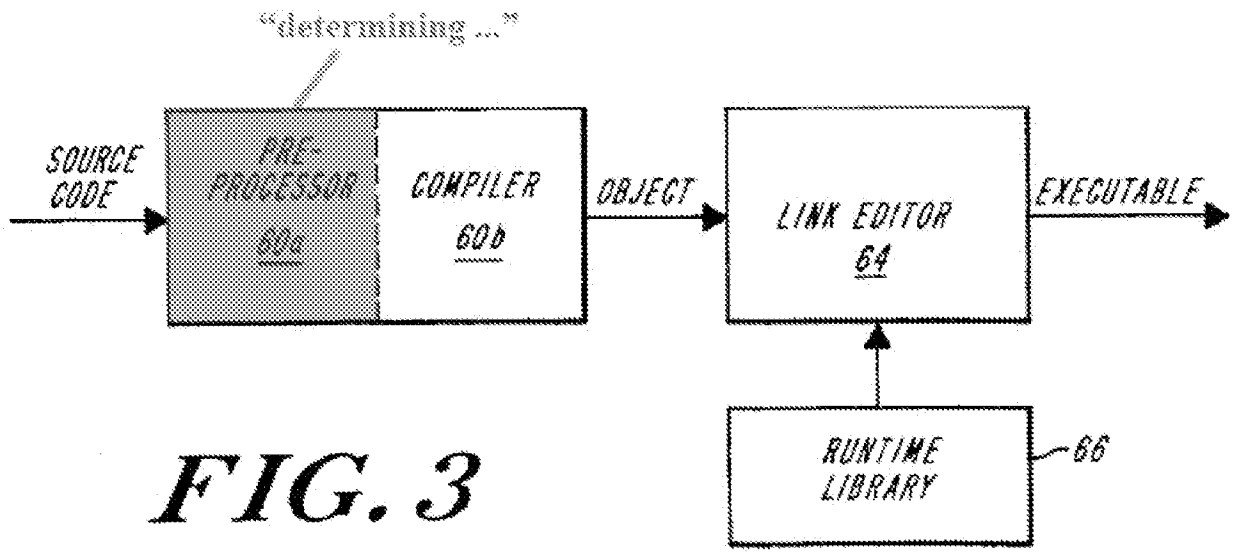
***Preexisting input file*:** In Reeve's system, "the input to preprocessor 60a is a *Fortran program source*[.]" *Id.*, 8:6-8. "The main role of preprocessor 60a is to put annotations into [the] Fortran program that enable parallel execution; that is, to insert tiling directives," and the "primary output is also a Fortran program—albeit a 'preprocessed' one—which includes the tiling directives. That second program is a valid and correct Fortran program, which when executed serially computes the same results as the original program." *Id.*, 8:1-12.

Accordingly, Reeve discloses, suggests, or renders obvious, element [1.pre].

b. [1.1]: "*(a) automatically determining file allocation and logically subdividing records of said input file into a plurality of partitions;*"

Reeve discloses, suggests, or at least render obvious, this element. EX1003, ¶¶ 132-140. Specifically, Reeve discloses *automatically determining* (automatic tiling process) *file allocation* (performing a data dependence analysis of loops in the input Fortran program) *and logically subdividing records* (indexed loops) *of said input file* (input Fortran program) *into a plurality of partitions* (loops that can/cannot be tiled). EX1003, ¶ 132.

[6] Reeve alternatively refers to the parallel processing cells as "cells", "processors", or "parallel processors", as well as "threads" or "pthreads" when executing software. *See, e.g.,* EX1006, 1:31-45, 2:3-18, 3:7-4:21, 5:33-35, 5:53-63, 16:56-17:45, 19:5-25, 22:44-26:36.



Reeve, FIG. 3*

Figure 3 of Reeve shows "a preferred arrangement of software modules utilized in digital data processor for parallelization and execution of software programs including iterative sequences. A compilation system 60 translates source code input into object code." EX1006, 7:14-18. As highlighted in the Figure, Reeve's pre-processor 60a performs the *automatically determining* step through its automatic tiling process. EX1003, ¶ 132.

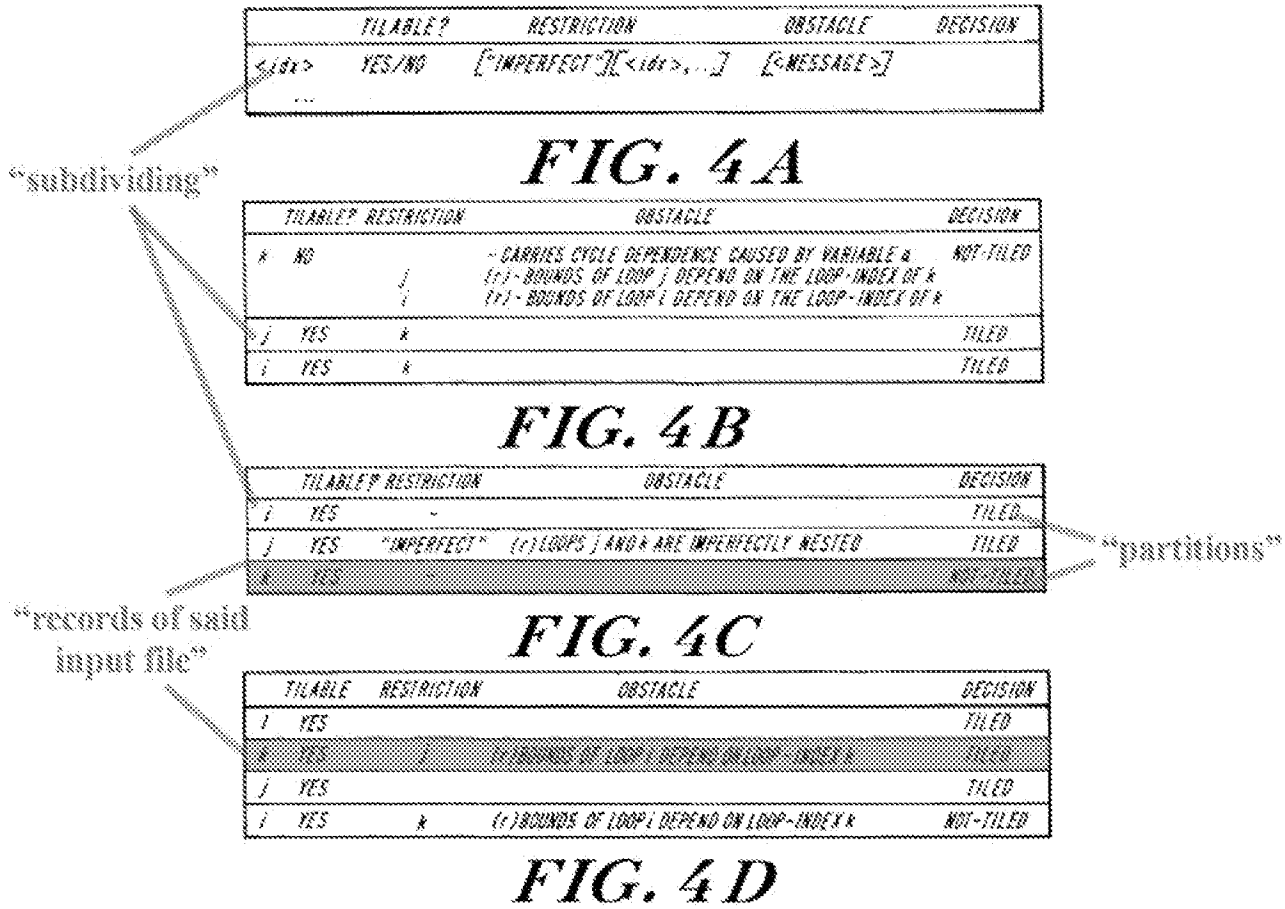
Automatically determining: Reeve's "automatic tiling" process, in which "the preprocessor 60a output[s] tiling directives," corresponds to the claimed *automatically determining* step. EX1006, 9:37-45; *see also* EX1003, ¶ 133.. In this tiling process, "[t]he main role of preprocessor 60a is to put annotations into a Fortran program that enable parallel execution; that is, to insert tiling directives. Preprocessor 60a also optionally performs some transformations of the code, such as loop interchanges, either to enable tiling or to optimize." *Id.*, 8:1-5.

***Determining file allocation*:** Reeve's tiling process determines file allocation, as claimed, by analyzing the input Fortran file to identify loops in the program that can be tiled and that cannot be tiled. EX1003, ¶ 134. Specifically, in Reeve's tiling process, "[t]he preprocessor 60a performs dependence analysis on the iterative sequence," and "data dependence is carried by loops." EX1006, 9:56-10:3. According to Reeve, some loops cannot be tiled because "a dependence is carried by [the] loop, [which] prevents tiling of that loop, while other loop(s) in the same loopnest can still be tiled." *Id.*, 10:4-8. Reeve identifies examples of data dependence in loops that present obstacles to tiling. For example, "some statements are not tilable," such as "a goto out of the

loopnest and a subroutine call." *Id.*, 10:8-11. "Another obstacle is a loop that is imperfectly nested," which "occurs where there are statements (other than DO's) between the loops in the loopnest." *Id.*, 10:37-43. Yet another example "is where the bound(s) of a loop depend on the index of an outer loop," which "implies a restriction that those two loops are mutually exclusive for tiling." *Id.*, 10:44-47.

Therefore, Reeve's preprocessor 60a examines the input Fortran program to identify loops whose degree of data dependence allows for tiling. EX1003, ¶ 135. The preprocessor "collects the information about the loopnest, tiling obstacles, etc.," and, "[t]hen, it takes the decision about which indices to tile." *Id.*, 11:9-14. Reeve's process of analyzing the input program, to determine dependence among loops that present obstacles to tiling, discloses the claimed *determining file allocation*. EX1003, ¶ 135.

***Logically subdividing records of said input file into a plurality of partitions*:** Reeve discloses this aspect by, after completing the dependence analysis, pre-processor 60a creating an indexed loop table that partitions the loops into those loops that can be tiled and those that cannot be tiled. EX1003, ¶ 136. Specifically, "[b]ased on its analysis of the loopnest (which-among other things-finds out all the tiling obstacles), preprocessor 60a tiles the loop while avoiding the tiling obstacles. In so doing, it produces a loop table which shows the tiling obstacles and the tiling decisions which are based on them." EX1006, 10:50-55. As shown in Figures 4A-D of Reeve below, the loop table includes an indexed entry for each loop (*record*) in the program (*input file*) that indicates a decision of whether the loop can or cannot be tiled:



Reeve, FIGS. 4A-4D*

Reeve explains that "preprocessor 60a creates the loop table in two steps. First, it collects the information about the loopnest, tiling obstacles, etc. Then, it [m]akes the decision about which indices to tile ...." EX1006, 11:9-16. Specifically, the loop table includes: (1) a "'tilable?' field" that "indicates whether there is a tiling obstacle which prevents this particular loop from being tiled, regardless of whether other loops are tiled or not"; (2) a "'restriction' field" that "notes which other loops in the loopnest might be affected by tiling this loop"; and (3) a "tiling-decision field" indicating the preprocessor's decision whether to tile the loop/index "based upon the information in the tilable? and restrictions fields." *Id.*, 11:15-50.

The preprocessor 60a then "starts from the first row in the loop table and moves down, tiling as much as possible, while taking care to respect any restrictions." *Id.*, 11:36-40. To "tile" a loop, Reeve's preprocessor operates to "insert tiling directives" into the input Fortran program by "put[ting] annotations into [the] Fortran program that enable parallel execution," creating a second Fortran program for the complier to execute. *Id.*, 8:1-5 *see also id.*, 11:1-2. As Reeve explains, the

tiling directives are "code" that includes tiling arguments ("tiling-args") with tiling parameters ("tiling-params"), and "which when executed serially computes the same results as the original program." *Id.*, 8:1-9:37. Accordingly, by *subdividing* the loops (*records*) of the input Fortran program (*input file*) into loops that can be tiled and loops that cannot be tiled (*partitions*), and tiling those that are tileable, Reeve discloses *logically subdividing records of said input file*, as claimed. EX1003, ¶ 137.

Second, Reeve further discloses the claimed *subdividing* by additionally grouping the tiles into tile families (*partitions*). *Id.*, ¶ 138. Reeve explains that "[t]he tiles are the basic entities which can be executed in parallel," and "[a] group of tiles which construct a loopnest is called tile-family. Numerous processors can execute the same loopnest, each one of them working on a separate tile simultaneously." EX1006, 15:49-55. Reeve's process "divide[s] the work among the pthreads [processor threads/cells] so that all the tiles which comprise the tile-family will be executed correctly and efficiently." *Id.*, 23:46-48. Accordingly, Reeve separately discloses *logically subdividing records of said input file*, as claimed, by subdividing the tiled loops (*records*) of the input Fortran program (*input file*) into tile families (*partitions*).

Records: As noted above, Reeve's loops correspond to the claimed *records of said input file* (input Fortran program file). EX1003, ¶ 139. This is consistent with the '582 Patent's disclosure regarding the claimed *records*. Other than indicating that *records* are elements of the input file that are distributed to, and processed by, the parallel Sub-Task processors, the '582 Patent provides little detail around the meaning of the term *record*. *See, e.g.*, EX1001, 5:27-6:40. Like the '582 Patent's *records*, Reeve's program "typically includes iterative sequences or 'loops'"—elements of the larger program file—that are allocated among the various parallel thread processors. EX1006, 7:19-27. EX1003, ¶ 139. Accordingly, Reeve's loops in the input Fortran program disclose the claimed *records of said input file*. EX1003, ¶ 139.

Accordingly, *Reeve* discloses, suggests, or renders obvious element [1.1].

c. [1.2]: "*(b) distributing descriptions of all of said partitions to each of a plurality of subtask processors*"

Reeve discloses, suggests, or at least renders obvious, this element. EX1003, ¶¶ 141-146. Specifically, Reeve discloses *distributing* (outputting to a central data structure called the Master Control Block (MCB)) *descriptions* (information regarding the tiles/tile families) *of all of said partitions to each of a plurality of subtask processors* (processor threads). EX1003, ¶ 141.

Distributing descriptions of all of said partitions the each of the plurality of subtask processors: Reeve discloses this aspect in at least two ways: (1) Reeve provides to all processor cells a master control block (MCB) containing the tiling information; and (2) Reeve provides to all processor cells a marker data structure that the processor cells use to claim tiles for execution. EX1003, ¶ 142.

Regarding the MCB, Reeve provides "a central datastructure called the MCB (Master Control Block) which is visible to all the threads." EX1006, 29:17-21. "The MCB holds the information regarding the tiles. The first section holds information about the tile-family. The second section holds information about each dimension of the tile family (order)--hence, it contains arrays(d) of fields, where d=1 . . . order." *Id.*, 29:28-33 *see also id.*, 29:24-30:14 (describing the content of the MCB). As highlighted in Reeve's MCB table below, for example, the MCB's tile description corresponds to the claimed *description of all of said partitions*:

MCB - Master Control Block

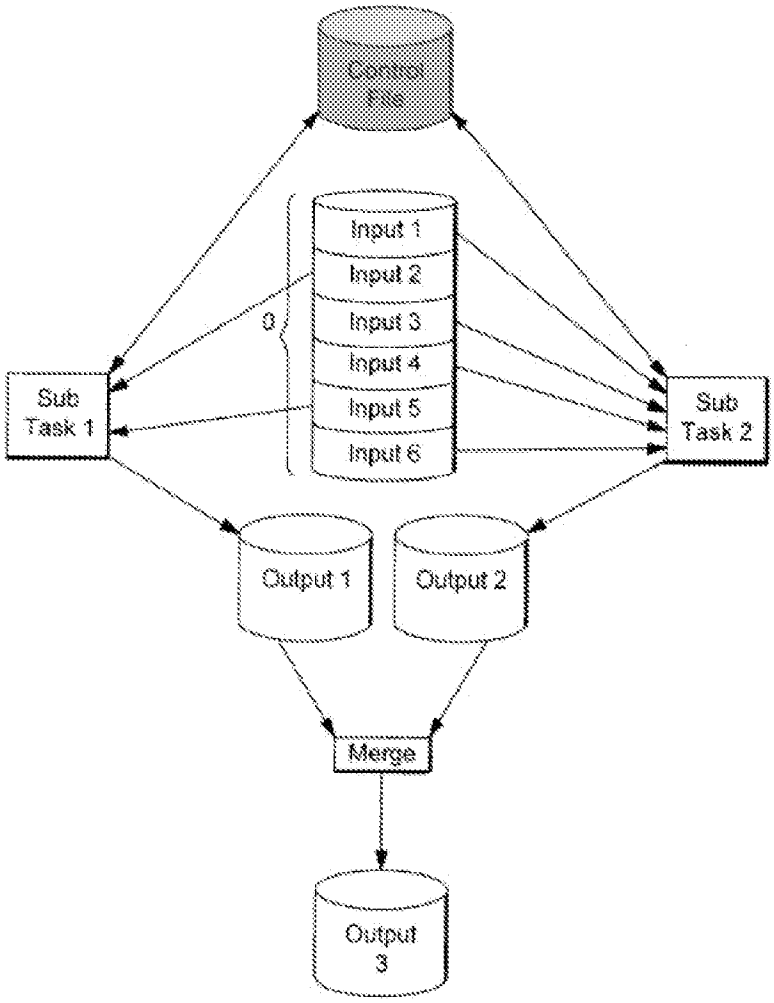
"descriptions of all of said partitions"

	Field	Description
		Family Section:
I	tile_efp	the FP (Frame Pointer) of the parent routine
C	tile_start	points to the text of the tile (in fact, CP of the tile)
C	order	as specified by tile-family-parameters
C	affinity	as specified by tile-family-parameters
C	dependency	as specified by tile-family-parameters
C	code_size	as computed by the compiler (back-end)
I	tile_strategy	code describing strategy to cover the tile family.
		Order Section:
I	tile_size (d)	# number of iterations in this dimension
C/I	loop-low-bound(d)	as specified by tile-family-parameters
C/I	loop-high-bound(d)	as specified by tile-family-parameters
C/I	loop-stride(d)	as specified by tile-family-parameters

Reeve, 29:40-65*

Regarding the marker structure, Reeve explains that processor cells "update a marker data structure to get the next available tile. This marker data structure is held in a lock." EX1006, 42:29-31. Reeve describes the process by which the processor cells access the marker data structure and lock tiles for execution. *See id.*, 42:28-55.

This is similar to how the '582 Patent makes the partition descriptions available to the subtask processors. EX1003, ¶ 145. Specifically, as described in the specification, the '582 Patent provides a central "control file"—in a similar role to Reeve's MCB and/or marker data structure—and "[e]ach Sub Task **finds the subdivision of the input in the control file 4**." EX1001, 3:67-4:1.



Reeve, FIG. 4*

Thus, Reeve's central control file 4 storing the partition information is visible to, and accessible by, the subtasks processors in the same way as Reeve's MCB and/or marker data structure. EX1003, ¶ 145.

Accordingly, by making the MCB and/or marker data structure (containing the claimed tile *descriptions*) accessible to the processor cells, Reeve discloses distributing the claimed descriptions *to each of the plurality of subtask processors*, as claimed. EX1003, ¶ 146.

d. [1.3]: *"c) simultaneously executing at least a respective one of the subtasks of the computer-executable process in each of at least some of said processors on a respective one of the partitions with each subtask reading and processing the respective partition so as to process the respective partition and produce respective subtask output and;"*

Reeve discloses, suggests, or at least renders obvious, this element. EX1003, ¶¶ 147-152. Specifically, Reeve's thread processors *simultaneously execute* (via the GRAB Strategy) *at least a respective one of the subtasks* (tiles/tile families) *of the computer-executable process* (parallel processing of an input Fortran program) *in each of at least some of said processors* (thread processors) *on a respective one of the partitions* (tiles) *with each subtask* (parallel processing thread) *reading and processing the respective partition so as to process the respective partition and produce respective subtask output* (execute the tile over the Fortran program). *Id.*, ¶ 147.

***Simultaneously executing at least a respective one of the subtasks of the computer-executable process in each of at least some of said processors on a respective one of the partitions*:** Reeve discloses a load-balancing "GRAB strategy" in which the thread processors simultaneously "grab," lock, and execute sets of one or more tiles until all the tiles of the program have been executed. *Id.*, ¶ 148. This is a passive load-sharing strategy, with no predetermined assignment of tiles to processors, because the processors grab the tiles for execution on "a first-come, first served basis." Specifically, regarding the characteristics of the GRAB strategy, Reeve explains:

> A grab strategy too divides the iteration space such that there are more tiles than available processors. The resultant tiles are assigned to requesting processors on a first-come-first-serve basis.

EX1006, 4:1-7.

> When using the GRAB strategy, Runtime Library 66 lets the scheduler handle all thread-processor bindings, load balancing, affinity, etc.

Id., 22:56-58.

> GRAB strategy
>
> This strategy issues tiles on a first-come, first-serve basis to the thread group. Each thread must obtain a common lock to access a tile for execution. This strategy is designed to load balance between the threads of the thread group.
>
> Note that this strategy does not consider data affinity considerations when assigning tiles to processors, and that it would not be a good strategy to choose in conjunction with affinity regions.

Id., 26:14-22.

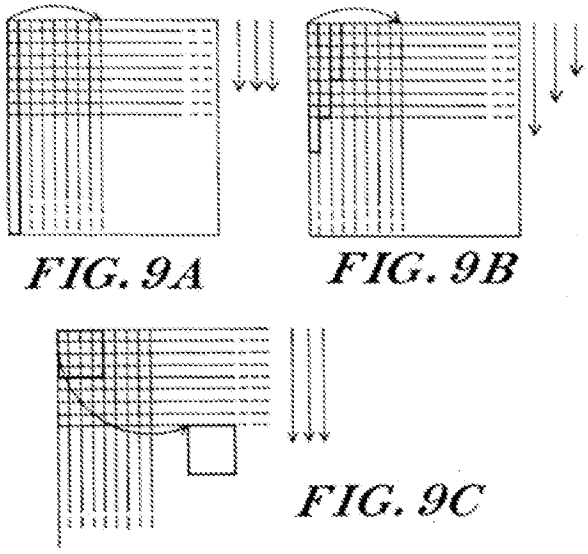
> The GRAB tiling strategy, is dynamic and will keep each thread busy as much as possible, but will probably cause data migration to occur more frequently. In addition to affinity loss problems is the additional synchronization overhead from the locking required to access a new tile.

Id., 26:32-36.

> 6.1.2 Locking during the grab strategy (critical section)
>
> The grab strategy requires all member threads to update a marker data structure to get the next available tile. This marker data structure is held in a lock.

Id., 42:28-31.

So as to process the respective partition and produce respective subtask output: As explained above, in Reeve's GRAB Strategy, the subtask processors grab and execute (*process*) tiles flagged as tileable (*respective partition*). *See, e.g.*, EX1006, 4:1-7, 26:14-22, 22:56-58, 26:32-36, 42:28-31; EX1003, ¶ 149. They do this until the entire tiled "iteration space" has been fully executed. *See, e.g.*, EX1006, 2:18-42, 3:11-24, 4:1-7. Each processor produces *respective subtask output* by "execut[ing] the tiled sequence over that tile." *Id.*, 2:25-27; *see also id.*, 50:40-41 ("executing said tiled sequence over the corresponding



tile"). This output has "the same results as the original program." *Id.*, 8:8-11; *see also id.*, 19:38-42 ("the same result as by running the same program serially").

Figures 9A-9C of Reeve, reproduced above, illustrate different techniques to execute a tiled sequence. Figure 9A, in particular, depicts a 2D iteration space tiled in two dimensions, where "the number of available processors is smaller than the number of columns" and "[t]here is no dependency." EX1006, 26:51-53. This configuration reflects the approach used in the GRAB strategy, which "divides the iteration space such that there are more tiles than available processors," and "[t]he resultant tiles are assigned to requesting processors on a first-come-first-serve basis." *Id.*, 4:1-7.

Accordingly, Reeve discloses, suggests, and/or renders obvious implementing the GRAB strategy—at least in situations where the tiles or tiled spaces exhibit low dependency or "affinity." *See, e.g., id.*, 26:14-22. According to Reeve, when used appropriately, "[t]he GRAB tiling strategy, is dynamic and will keep each thread busy as much as possible[.]" *Id.*, 26:32-36; *see also* EX1003,¶ 151. This is because the GRAB strategy passively "lets the scheduler handle all thread-processor bindings, load balancing, affinity, etc." EX1006, 22:56-58.

Accordingly, Reeve discloses, suggests, or renders obvious element [1.3].

e. [1.4] "*d) thereafter repeating step (c) in at least some of the subtask processors each with another unprocessed partition on a first-come/first-served basis; and*"

Reeve discloses, suggests, or renders obvious this element. EX1003, ¶¶ 153-154. Specifically, Reeve's GRAB *strategy thereafter repeat[es] step (c) in at least some of the subtask processors* (processing cells) *each with another unprocessed partition* (remaining unprocessed tile) *on a first-come/first-served basis* (via a first-come, first-serve basis). As explained above, in Reeve's GRAB strategy, upon completing execution of a tile, each processor returns to grab, lock, and execute an unprocessed tile until the entire tiled space has been processed. *See, e.g.*, EX1006, 4:1-7, 22:56-58, 26:14-22, 26:32-36, 42:28-31. Reeve expressly states that "[t]his strategy issues tiles on a first-come, first-serve basis to the thread group." *Id.*, 26:14-22; *see also id.*, 4:1-7 ("A grab strategy ... divides the iteration space such that there are more tiles than available processors. The resultant tiles are assigned to requesting processors on a first-come-first-serve basis."); EX1003, ¶ 153.

Accordingly, Reeve discloses, suggests, or renders obvious element [1.4].

f. [1.5] *"(e) generating at least one output combining all of the subtask outputs and reflecting the processing of all of said subtasks."*

Reeve discloses, suggests, or renders obvious this element. EX1003, ¶¶ 155-157. Specifically, Reeve discloses *generating at least one output combining* (the executed tiled space after completing execution) *all of the subtask outputs* (output resulting from executing the individual tiles over the iteration space) *and reflecting the processing of all of said subtasks*.

In Reeve, after the tiled iteration space is divided, the processor cells collectively execute the program by "execut[ing] the tiled sequence over that tile" until the entire tiled space has been completed. EX1006, 2:25–27. The resulting, combined output yields "the same results as the original program." *Id.*, 8:9–11; *see also id.*, 19:38–42 ("the same result as by running the same program serially"). In describing the GRAB strategy, Reeve explains that there are more tiles than processor cells, meaning that all processor cells participate in the parallel execution of the tiled program: "A grab strategy too divides the iteration space such that there are more tiles than available processors. The resultant tiles are assigned to requesting processors on a first-come first-serve basis." *Id.*, 4:1–4; *see also* EX1003, ¶ 156.Because all processor cells participate in executing tiles in the GRAB strategy, the resulting *output combin[es] all of the subtask outputs* (processor cell outputs) *and reflecting the processing of all of said subtasks* (processor cells), as claimed. EX1003, ¶ 156.

Accordingly, Reeve discloses, suggests, or renders obvious element [1.5].

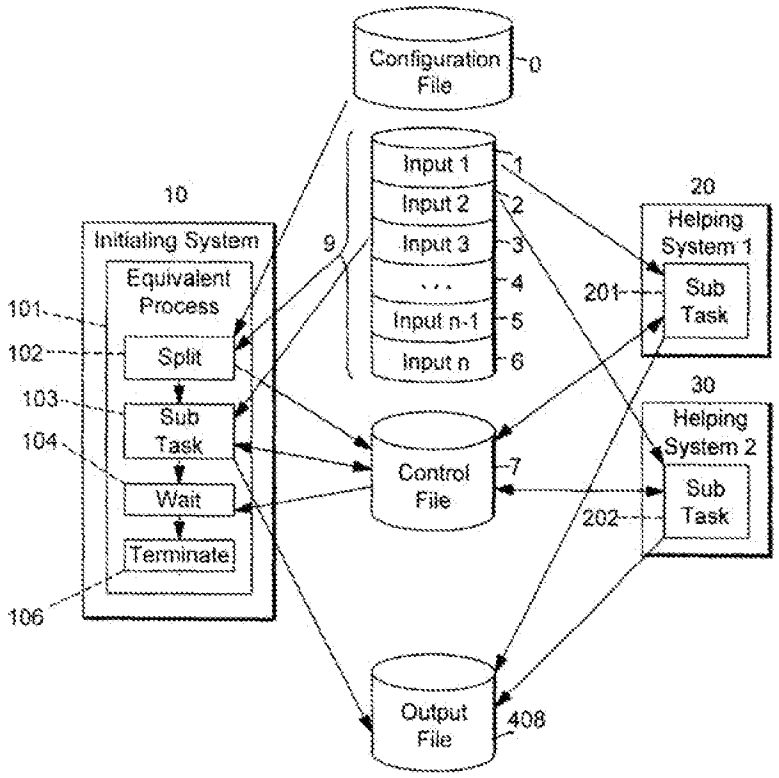
2. Dependent Claim 2

a. [2.1]: "*The method defined in claim 1 wherein the automatic determination of file allocation and logical subdivision of records of said input file into said plurality of partitions in step (a) and the distribution of the description of all of said partitions in step (b) is carried out with at least one further processor in addition to the subtask processors.*

As explained above, Reeve renders claim 1 unpatentable. *Supra* Section VI.B.1. Moreover, Reeve discloses, suggests, or renders obvious the additional elements of dependent claim 2. Specifically, Reeve discloses that each of steps (a) and (b) recited in claim 1 "*is carried out with at least one further processor in addition to the subtask processors*"—Reeve's preprocessor 60a. EX1003, ¶¶ 158-160.

As explained above for elements [1.1] and [1.2]. Reeve's preprocessor 60a performs the claimed steps (a) and (b). *Supra* Sections VI.B.1.b, VI.B.1.c. Part of Reeve's compilation system 60 shown in Figure 3, preprocessor 60a that performs the tiling process is a separate and distinct processor from the cell processors, shown in Figure 2, that use the GRAB strategy to execute individual tiles. *Compare* EX1006, 5:53-6:27, FIG. 2 *with id.*, 7:14-47, FIG. 6.

Reeve's configuration is similar to that of the '582 Patent which has an initiating system 10 (*at least one further processor*) that creates the partitions and multiple helping systems 20, 30 (*subtask processors*) that perform the tasks:



'582 Patent, FIG. 3

Accordingly, Reeve discloses, suggests, or renders obvious the elements of claim 2. EX1003, ¶¶ 159-160.

3. Dependent Claims 3-5

a. [3.1]: "*The method defined in claim 1, further comprising the step of merging said subtask outputs in step (e).*"

[4.1] "*The method defined in claim 1 wherein the output in step (e) is a succession of outputs from said subtasks in a one-to-one correspondence with said records of said input file.*"

[5.1] "*The method defined in claim 1 wherein the output in step (e) is an accumulation of output records from said subtasks in an arbitrary order.*"

As explained above, Reeve renders claim 1 unpatentable. *Supra* Section VI.B.1. Moreover, Reeve discloses, suggests, or renders obvious the additional elements of dependent claims 3, 4, and 5. EX1003, ¶¶ 161-164. As explained above for *step (e)* of independent claim 1, Reeve's thread processors produce respective outputs by "execut[ing] the tiled sequence over that tile" until the entire tiled space has been executed. EX1006, 2:25-27. And the resulting, combined output produces "the same results as the original program." *Id.*, 8:9–11; *see also id.*, 19:38–42 ("the same result as by running the same program serially"). That is, Reeve's tiling execution strategy combines or *merges* the *subtask outputs* of the individual processors—as recited in element [3.1]—as a single, executed program. EX1003, ¶ 161.

Regarding element [4.1], Reeve's tiled program has a *one-to-one correspondence* with the original program because every loop in the original program is executed in the final program—either separately or as part of a tile. EX1003, ¶ 162. This is why Reeve's tiled program has the has "the same results as the original program"—i.e., "the same result as by running the same program serially." EX1006, 8:9–11, 19:38–42. Thus, the sequential execution of the tiles by Reeve's thread processors during the GRAB strategy has *a one-to-one correspondence with said records* (loops) *of said input file* (input Fortran program). EX1003, ¶ 162.

Regarding element [5.1], Reeve explains that "[t]he GRAB tiling strategy ... is dynamic and will keep each thread busy as much as possible," but Reeve's thread processors do not necessarily finish executing the tiles at the same time or in any specific order. EX1006, 26:32-36. Indeed, "[t]he resultant tiles are assigned to requesting processors on a first-come-first-serve basis"—as the processors become ready for more work—"to load balance between the threads of the thread group." *Id.*, 4:1-7; *see also id.*, 26:14-22. By completing execution of the tiles in no prescribed order—but on a first-come, first-served basis—Reeve discloses that the thread

processors combined output *is an accumulation of output records* (executed) *from said subtasks* (processors) *in an arbitrary order*. EX1003, ¶ 163.

Accordingly, Reeve renders claims 3-5 unpatentable.

4. Dependent Claims 6 and 7

a. [6.1]: "*The method defined in claim 1 wherein said input file resides on a storage area network and is derived therefrom.*"

[7.1]: "*The method defined in claim 1 wherein said input file resides on a network-attached storage and is derived therefrom.*"

As explained above, Reeve renders claim 1 unpatentable. *Supra* Section VI.B.1. Moreover, Reeve discloses, suggests, or renders obvious the additional elements of dependent claims 6 and 7. EX1003, ¶¶ 165-168. Reeve discloses, suggests, or renders obvious these elements in two ways.

First, Reeve does not disclose where the input source Fortran program originates, but Reeve discloses that its devices may have various types of storage. *See, e.g.*, EX1006, 5:53-6:27, FIG. 2 (memory elements 40, stores 54, etc.). A POSITA would have understood that the input source Fortran program must be stored in some storage device and have found the claimed *storage area network* and *network-attached storage* as obvious options. EX1003, ¶ 166. Accordingly, to the extent that Reeve does not expressly disclose or suggest the claimed *storage area network* or *network-attached storage*, Reeve renders them obvious. *Id.*

Second, as shown in Figure 3 of Reeve, pre-processor 60a receives, and therefore necessarily stores or obviously must store, the source Fortran program. EX1006, 7:14-27, FIG. 3; *see also id.*, 7:28-31 ("The compilation system 60 includes a preprocessor 60a and a compiler 60b. The preprocessor 60a preforms preliminary analysis of iterative sequences in the source code to determine the dependency directions thereof"); EX1003, ¶ 167. Moreover, Reeve teaches that compilation system 60 outputs a tiled Fortran program for execution by Reeve's processing cells, and Figure 2 of Reeve shows that the processing cells communicate over network 46. *See* EX1006, 5:58-63 ("Network 46, representing the aforementioned level segments and routing cells, transfers information packets (passed to the network 46 over buses 48A, 48B and 48C) between the illustrated processing cells 42A-42C."), FIG. 2. Based on this, a POSITA would have understood or found obvious that compilation system 60 provides the processing cells with access to the output Fortran program over network 46. EX1003, ¶ 167. For this additional reason, Reeve discloses, suggests, or renders obvious the claimed *network storage area network* and *network-attached storage* to store the Fortran file. *See also* EX1019 (disclosing distributed memory models,

network-accessible storage, and parallel file systems that support concurrent data access across processor nodes, consistent with SAN and NAS architectures).

Accordingly, Reeve discloses, suggests, or renders obvious claims 6 and 7.

5. Dependent Claims 8-11

a. [8.1]: "*The method defined in claim 1 wherein said computer-executable process is a sort process.*"

[9.1]: "*The method defined in claim 1 wherein said computer-executable process is a statistical analysis process.*"

[10.1]: "*The method defined in claim 1 wherein said computer-executable process is a report-creating process.*"

[11.1] *"The method defined in claim 1 wherein said computer-executable process includes a database query.*"

During original prosecution, the Examiner rejected claims 8-11 as unpatentable over the Kitsuregawa reference because each merely recites an intended use of the claimed method without further meaningfully limiting the claimed method: use in a sort process, a statistical analysis process, a report-creating process, and a database query, respectively. *See* EX1002, 59. The Applicant did not dispute the Examiner's rejection or characterization of claims 8-11. Indeed, the Examiner was correct. EX1003, ¶¶ 169-173.

The preamble of independent claim 1 calls for a method of affecting "a computer-executable process" on a preexisting input file—i.e., an intended use of the claimed method on a "computer-executable process." Dependent claims 8-11 simply add further preamble labeling that process a "sort" process, a "statistical analysis" process, a "report-creating" process, and a "database query," respectively. But neither the body of independent claim 1 nor the bodies of dependent claims 8-11 recites any further process steps or structural elements related to such use in those processes—or any other type of computer-executable process. Because "the body of [the] claim fully and intrinsically sets forth all of the limitations of the claimed invention, and the preamble merely states ... the purpose or intended use of the invention, rather than any distinct definition of any of the claimed invention's limitations," "the preamble is not considered a limitation and is of no significance to claim construction." MPEP § 2111.02(II). Applicant's failure to rely on the intended uses recited in claims 8-11 to distinguish prior art confirms claims 8-10 are not entitled to patentable weight because "[w]ithout such reliance, ... a preamble generally is not limiting when the claim body describes a structurally complete invention such that deletion of the

preamble phrase does not affect the structure or steps of the claimed invention." *Id.* (citing *Catalina Mktg. Int'l v. Coolsavings.com, Inc.*, 289 F.3d 801, 808 (Fed. Cir. 2002).

Indeed, as Dr. Weissman explains, the claimed technique could be used on many different types of computer-implemented processes besides the examples mentioned in claims 8-11. EX1003, ¶ 171. And it was well known in the art to use parallel processing techniques in each of the applications recited in claims 8-11:

- **Claim 8:** *See, e.g.*, EX1005, EX1011, EX1012, EX1015, EX1017, EX1022, and (disclosing different types of parallel sorting techniques).
- **Claim 9:** *See, e.g.,* EX1016 and EX1018 (disclosing parallel statistical analysis methods such as least median of squares and isotonic regression).
- **Claim 10:** *See, e.g.*, EX1022 (disclosing parallel report generation using distributed file systems).
- **Claim 11:** *See, e.g.*, EX1013 (disclosing parallel database query execution).

See EX1003, ¶ 171.

Consistent with this, the specification of the '582 Patent teaches that the claimed method generally applies to "the field of sharing data and workload" among computer systems, highlighting that "[s]ort applications, statistical analysis batch applications and report writing applications and database queries are [mere] *examples* of applications that can readily enjoy this invention." Nothing in the '582 Patent claims or specification suggests that the claimed method is limited to the four claimed potential uses and would exclude Reeve's application of distributed execution of tiled portions of a computer program in a multiprocessor system. EX1003, ¶ 172.

Accordingly, for the reasons above, claims 8-11 are not entitled to patentable weight, and Reeve renders these claims unpatentable.

6. Dependent Claim 12

a. [12.1]: "*The method defined in claim 2 wherein said one processor is part of a mainframe computer and the other processors are processors of at least one other computer.*"

Reeve renders claim 2 unpatentable. *Supra* Section VI.B.2. Moreover, Reeve discloses, suggests, or renders obvious the additional elements of dependent claim 12. EX1003, ¶¶ 174-178.

As explained above, claim 2 recites that the "*one [further] processor*" performing steps (a) and (b) of claim 1 is a processor in addition to the *subtask processors*, and Reeve's preprocessor

60a of Figure 3 corresponds to the *one processor* while Reeve's cell processors of Figure 2 correspond to the claimed *subtask processors*. *Supra* Section VI.B.2. Claim 12 adds (1) the claimed one processor is *a mainframe computer*; and (2) the subtask processors are those *of at least one other computer*.

Taking (2) first, Reeve describes the cell processors (Figure 2) as residing in different computers from the pre-processor 60 (Figure 3). *Compare* EX1006, 5:53-6:27, FIG. 2 *with id.*, 7:14-37, FIG. 3. Thus, Reeve discloses the aspect of claim 12 reciting that the *other processors are processors of at least one other computer*. EX1003, ¶ 176.

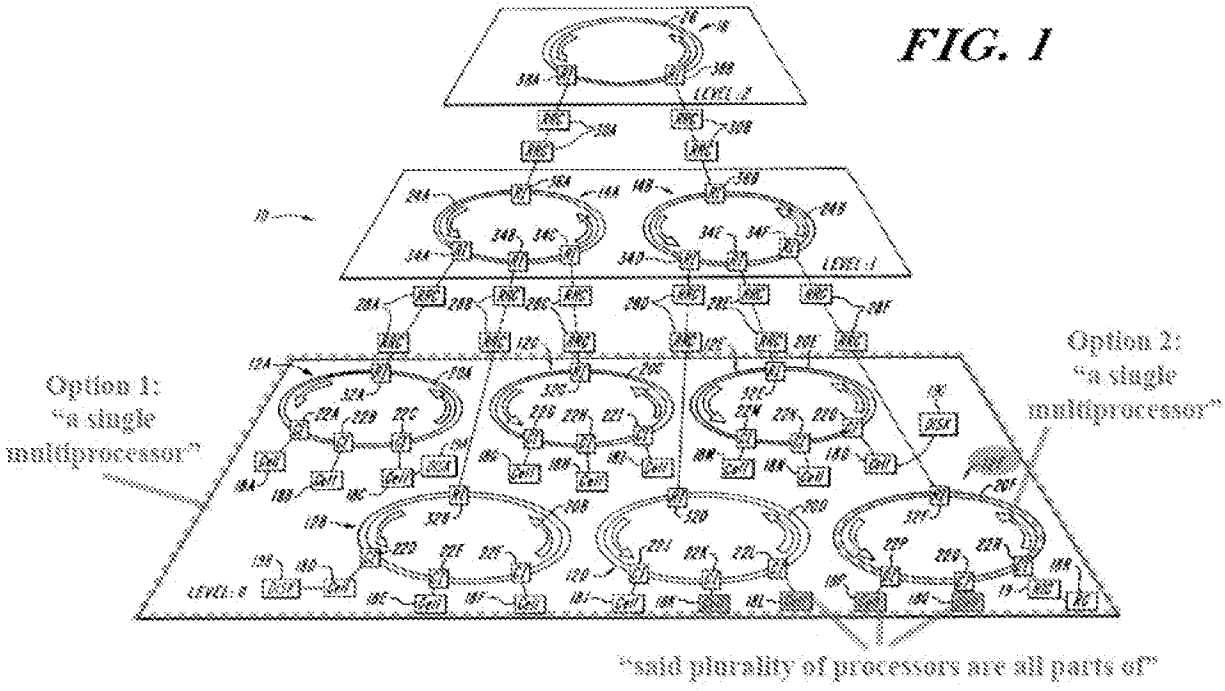
Regarding (1), a *mainframe* computer was a high-level, typically large and expensive computer designed to handle intensive computational tasks. EX1008, 6. Because of this, and as a POSITA was aware, the earliest computers used to manage parallel processing strategies were mainframe computers. EX1003, ¶ 177. Reeve's compilation computer system 60, which includes pre-processor 60a, performs computationally-intensive tasks including tiling and otherwise parallelizing the input program for execution by the individual cell processors. *See generally* EX1006 7:14-44:18. As other examples, EX1014 and EX1020 disclose parallel architectures in which a mainframe or master computer supervises or coordinates parallel processors in at least one other computer. Thus, while Reeve does not call the computer hosting pre-processor 60a *mainframe* computer, specifically, this would have been a well-known, practical configuration for Reeve's compilation system 60 for the same reasons discussed above for Ground 1. EX1003, ¶ 177. Needing some physical implementation for pre-processor 60a, therefore, a POSITA would have been motivated and found it obvious to use a *mainframe* computer for this purpose. *Id.*

Accordingly, Reeve renders claim 12 unpatentable.

7. Dependent Claim 13

a. [13.1] "*The method defined in claim 1 wherein said plurality of processors are all parts of a single multiprocessor.*"

Reeve renders claim 1 unpatentable. *Supra* Section VI.B.1. Moreover, Reeve discloses, suggests, or renders obvious the additional element of dependent claim 13. EX1003, ¶¶ 179-183. Specifically, in Reeve's system, *said plurality of processors* (cell processors 18A-Q) *are all parts of a single multiprocessor* (information transfer level 0 and/or a segment 12A-F thereof):



Reeve, FIG. 1*

As an initial matter, Reeve characterizes its system as a "multiprocessor system," as claimed. *See, e.g.*, EX1006, Abstract ("The improved apparatus and method have application to a variety of *multiprocessor systems,* including those which are massively parallel."), 1:34-37 ("A special class of these *multiprocessors*, referred to as parallel processors, ..."), 1:65-67 ("Yet another object is to provide a computer run-time mechanism for parallelizing and executing a computer program on a multiprocessor system."); *see also* EX1003, ¶ 180.

As highlighted above, Reeve discloses element [13.1] in two ways. First, Reeve's "multiprocessing system ... 10 includes three information transfer levels: level:0, level:1, and level:2." EX1006, 5:23-32. All of Reeve's individual cell processors 18 (claimed *plurality of processors*) are part of information transfer level 0, annotated above with a dashed border. *Id.*, 5:33-36 ("Each segment of level:0, i.e., segments 12A, 12B, . . . 12F, comprise a plurality of processing cells. For example, segment 12A includes cells 18A, 18B and 18C; segment 12B includes cells 18D, 18E and 18F; and so forth."); EX1003, ¶ 181. Because they are all part of the same information transfer level 0, Reeve's cells 18 *all parts of a single multiprocessor*, as claimed.

Second, within level 0, individual cell processors 18 are interconnected over segments 12A-12F. EX1006, 5:33-43. For example, as shown above, segment 12F interconnects cells 18P and 18Q. Processing cells 18 on a given segment 12 interact over network 46 (shown in Figure 2):

> FIG. 2 illustrates in greater detail processing cells and their interconnection within the processing system of FIG. 1. In the drawing, plural central processing units 40A, 40B and 40C are coupled, respectively, to associated memory elements 42A, 42B and 42C. Communications between the processing and memory units of each pair are carried along buses 44A, 44B and 44C, as shown. Network 46, representing the aforementioned level segments and routing cells, transfers information packets (passed to the network 46 over buses 48A, 48B and 48C) between the illustrated processing cells 42A-42C.

Id., 5:53-63. Accordingly, a given set of processing cells 18 interconnected and cooperating on the same segment 12—within level 0—are also *all parts of a single multiprocessor*, as claimed. EX1003, ¶ 182; *see also* EX1010, EX1011, EX1012, and EX1014 (additional prior art examples disclosing parallel processors operating within a single multiprocessor system architecture).

Accordingly, Reeve renders claim 13 unpatentable.

8. Dependent Claim 14

a. [14.1] "*The method defined in claim 1 wherein the automatic determination of file allocation and logical subdivision of records of said input file into said plurality of partitions in step (a) and the distribution of the descriptions of all of said partitions in step (b) is carried out with at least one processor, and said one processor and said plurality of processors are all parts of a single multiprocessor not including said one processor.*"

Reeve renders claim 1 unpatentable. *Supra* Section VI.B.1. Reeve also discloses, suggests, or renders obvious the additional element of dependent claim 14. As an initial matter, claim 14 is inoperable and/or nonsensical because it simultaneously requires the one processor to *both* be part of *and* not be part of a single microprocessor, which is logically impossible. EX1003, ¶¶ 184-186.

To the extent claim 14 is not indefinite and Applicant intended it to recite that the *single multiprocessor* that includes the *plurality of processors* does not also include the *one processor* that performs the claimed steps (a) and (b), Reeve discloses it. EX1003, ¶ 185. As explained above for claim 13, Reeve's cell processors 18 (claimed *plurality of processors*) *are all parts of a single multiprocessor*, as claimed, because they are interconnected within level 0. Separately, sets of cell processors 18 interconnected on the same segment 12 on level 0 of Reeve's system 10 *are all parts of a single multiprocessor*, as claimed. *Supra* Section VI.B.7. Moreover, as explained above for

claim 1, Reeve's compilation system 60 that includes pre-processor 60a (FIG. 3) performs the claimed steps (a) and (b). *Supra* Sections VI.B.1.b, VI.B.1.c But the compilation system 60 does not belong to level 0 and/or a segment 12 along with the cell processors 18. *See* EX1006, 5:23-53, FIG .1; EX1003, ¶ 185. Thus, Reeve's *single multiprocessor* (level 0 / segment 12) that includes the *plurality of processors* (processing cells 18) does not also include the *one processor* (compilation system 60 including pre-processor 60a) that performs the claimed steps (a) and (b). EX1003, ¶ 185.

For at least the reasons above, Reeve alone anticipates and/or renders obvious Challenged Claims 1-14.

VII. DISCLOSURE OF CONCURRENT LITIGATION, REEXAMINATION, AND RELATED PROCEEDINGS

Based on the information available to Requester, the '582 Patent is the subject of three pending District Court litigations and PTAB proceedings, as listed in the table below. Requester is unaware of any prior reexaminations or other post-grant proceedings involving the '582 Patent.

Title	Case No.	Court	Filed	Status
Intellectual Ventures I LLC et al v. American Airlines, Inc.	14-24-cv-00980	E.D Tex.	Nov. 2, 2024	Pending
Intellectual Ventures I LLC et al v. Southwest Airlines Co.	7-24-cv-00277	W.D. Tex.	Nov. 2, 2024	Pending
American Airlines, Inc. et al v. Intellectual Ventures I LLC	IPR2025-00785	P.T.A.B	Apr. 10, 2025	Pending

VIII. CONCLUSION

Requester requests the issuance of an Office Action rejecting the Challenged Claims based on one or more of Grounds 1-3 detailed above. The Commissioner is hereby authorized to charge Deposit Account 12-0769 under Docket No. US022-0014RE the *Ex Parte* Reexamination fee of $12,600 under 37 C.F.R. § 1.20(c)(2). Requester believes no other fee is due with this submission; however, the Commissioner is hereby authorized to charge any fee deficiency or credit any over-payment to Deposit Account 12-0769.

Please direct all correspondence in this matter to the undersigned.

Respectfully submitted,

Dated: June 20, 2025

By: */James D. Stein/*

James D. Stein (Reg. No. 63,782)
Lee & Hayes, P.C.
75 14th Street NE, Suite 2500
Atlanta, Georgia 30309
james.stein@leehayes.com
P: 404-736-1918

Counsel for Requester Unified Patents, LLC

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re patent of Michael Rothschild	§	Attorney Docket No.: U022-0014RE
	§	
U.S. Patent 7,257,582	§	
	§	Customer No.: 29150
Issue Date: August 14, 2007	§	
	§	
Filing Date: February 27, 2003	§	
	§	
For: LOAD BALANCING WITH	§	
SHARED DATA	§	

Mail Stop "*Ex Parte* Reexam"
Attn: Central Reexamination Unit
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

CERTIFICATE OF SERVICE

Pursuant to 37 C.F.R. §§ 1.248 and 1.550 and MPEP § 2266.03, the undersigned attorney for the Requester certifies that a copy of the foregoing **REQUEST FOR *EX PARTE* REEXAMINATION OF U.S. PATENT NO. 7,257,582** and accompanying Exhibits EX1001-1026 were served by Priority Mail Express or equivalent at the following address of record for the subject patent:

86636 - BRUNDIDGE & STANGER, P.C.
1925 BALLENGER AVENUE, STE. 560
ALEXANDRIA, VA 22314
UNITED STATES

An additional courtesy copy was served electronically on Patent Owner's litigation counsel:

Jonathan K. Waldrop
jwaldrop@kasowitz.com
Darcy L. Jones
djones@kasowitz.com
Marcus A. Barber
mbarber@kasowitz.com
John W. Downing
jdowning@kasowitz.com
Heather S. Kim
hkim@kasowitz.com

ThucMinh Nguyen
tnguyen@kasowitz.com
KASOWITZ BENSON TORRES LLP
333 Twin Dolphin Drive, Suite 200
Redwood Shores, California 94065

Respectfully submitted,

Dated: June 20, 2025

By: /*James D. Stein*/

James D. Stein (Reg. No. 63,782)
Lee & Hayes, P.C.
75 14th Street NE, Suite 2500
Atlanta, Georgia 30309
james.stein@leehayes.com
P: 404-736-1918

Counsel for Requester Unified Patents, LLC