(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) **World Intellectual Property Organization**
International Bureau

PCT

(43) **International Publication Date**
27 December 2007 (27.12.2007)

(10) **International Publication Number**
WO 2007/149228 A1

(51) **International Patent Classification:**
G06F 9/50 (2006.01)

(21) **International Application Number:** PCT/US2007/013452

(22) **International Filing Date:** 6 June 2007 (06.06.2007)

(25) **Filing Language:** English

(26) **Publication Language:** English

(30) **Priority Data:**
11/471,466 19 June 2006 (19.06.2006) US
11/546,072 10 October 2006 (10.10.2006) US

(71) **Applicant** *(for all designated States except US)*: **DISKEEPER CORPORATION** [US/US]; 7590 N. Glenoaks Blvd., Burbank, California 91504 (US).

(72) **Inventors; and**
(75) **Inventors/Applicants** *(for US only)*: **JENSEN, Craig** [US/US]; 4245 Mesa Vista Drive, La Canada, California 91011 (US). **STAFFER, Andrew** [CA/US]; 13270 Alta Vista Way, Sylmar, California 91342 (US). **THOMAS, Basil** [IN/US]; 14537 Willowgreen Lane, Sylmar, California 91342 (US). **CADRUVI, Richard** [US/US]; 530 Sonata Way #D, Simi Valley, California 93065 (US).

(74) **Agents: POMERENKE, Ronald, M.** et al.; 2055 Gateway Place, Suite 550, San Jose, California 945110 (US).

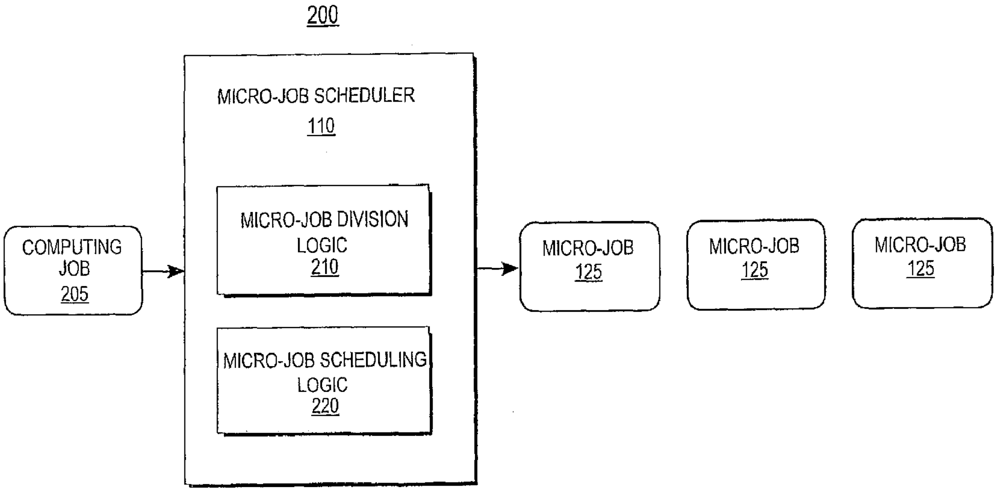
(81) **Designated States** *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, MT, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

[Continued on next page]

(54) Title: COMPUTER MICRO-JOBS



(57) **Abstract:** Computer micro-jobs are disclosed. A computer job is divided into micro-jobs. In one embodiment, the micro-jobs have a size that allows a particular micro-job to complete within an allotted time for which the particular micro-job owns a resource used to execute the micro-job. In one embodiment, the allotted time is a quantum. In one embodiment, an entire computer job is divided into micro-jobs and the computer job is then executed micro-job by micro-job until the entire computer job is complete. Each of the micro-jobs may complete its execution within its quantum, in one embodiment. In one embodiment, the execution of the micro-jobs is allocated to times when needed resources comply with one or more idleness criteria. A software program executed with micro-jobs may be run at all times while the computer is powered up without impacting the performance of other software programs running on the same computer system.

WO 2007/149228 A1

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

COMPUTER MICRO-JOBS

FIELD OF THE INVENTION

[0001] The present invention relates to executing software applications in a computer environment. In particular, embodiments of the present invention relate to dividing a computer or input-output job of an application into micro-jobs and executing the micro-jobs.

BACKGROUND

[0002] In many multi-tasking operating systems, processes are broken up into several threads. A thread is a piece of code executed by the operating system (O/S). The concept of multi-threading is to allow several pieces of code (or threads) in one process to run "concurrently." For example, if a word processor is running, the user can click on a "find menu item" to cause a pop-up box to appear. This pop-up can be moved and manipulated independent of the main word processor window. Thus, the pop-up does not render the main word processor window inactive. This is an example of two different threads running within the word processor process.

[0003] The concept of multi-tasking is similar to multi-threading in that it gives the appearance of multiple pieces of code executing at the same time on a single computer processor. A difference is that multi-tasking refers to more than one process running on the computer and multi-threading refers to more than one thread running within the same process as with the example above.

[0004] The appearance of more than one process or thread running concurrently is a result of a multi-tasking scheduler scheduling threads to run in very small time increments, which may be referred to as "quanta." A quantum is a time slice given to a thread during which time that thread owns a CPU resource. The length of a quantum is in the range of about 20 milliseconds to about 120 milliseconds on contemporary Operating Systems. The exact times may vary depending on the hardware on which the O/S is running. Further, the O/S can change the value of the quantum given to a particular thread. For example, if a thread does not complete during its first quantum, the O/S might increase or decrease the size of the quantum the next time the thread is scheduled to execute.

[0005] Due to the small length of a quantum compared to the human perception of time and by executing threads in a round robin fashion, it appears that threads run concurrently. Contemporary multi-tasking O/S schedulers add priorities to threads and various algorithms exist to optimally run the higher priority threads before the lower priority threads. However, all threads are presented to the O/S scheduler for immediate execution and the O/S scheduler to the best of its ability gets all the threads to complete their execution as fast as it can based on their priority.

[0006] However, a problem with scheduling in this manner is that computer performance is poorer than might be expected. Often, a process hesitates or even freezes. For example, a process that renders on a display screen based on user input is often unable to render as the user inputs data because another process is consuming too much processor time.

[0007] The approaches described in this section are approaches that could be pursued, but not necessarily approaches that have been previously conceived or pursued. Therefore, unless otherwise indicated, it should not be assumed that any of the approaches described in this section qualify as prior art merely by virtue of their inclusion in this section.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

[0009] FIG. 1 is a diagram of an architecture for executing micro-jobs, in accordance with an embodiment of the present invention.

[0010] FIG. 2 is a diagram micro-scheduler that divides a computer job into micro-jobs, in accordance with an embodiment of the present invention.

[0011] FIG. 3 is a flowchart illustrating steps of a process of dividing a computer job into micro-jobs, in accordance with an embodiment of the present invention.

[0012] FIG. 4 is a block diagram that illustrates a computer system upon which an embodiment of the invention may be implemented.

DETAILED DESCRIPTION

[0013] In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the present invention.

OVERVIEW

[0014] The majority of computers do not utilize all of their resource capacity 100% of the time. This is typically true even of computers that seemingly are in high use twenty-four hours a day, seven days a week, as well as computers that are only turned on for a portion of each day. Therefore, computer time and resources are wasted. For example, over a twenty-four hour period, a computer system that is used quite heavily, and which may have brief spikes in activity, may on average use only about five to twenty percent of its resources.

[0015] A method, system, and apparatus are disclosed herein to utilize these unused computer resources by dividing a computer job into micro-jobs. A micro-job may be a very small sliver of computer code (e.g., a relatively small number of instructions). In one embodiment, the size of a micro-job is such that it will finish execution within a time for which the micro-job was allocated a processing resource. For example, the number of operations in the micro-job is such that it will complete execution within a quantum, in one embodiment. A micro-job may be shorter in length than a quantum.

[0016] In one embodiment, an entire computer job is divided into micro-jobs and the computer job is then executed micro-job by micro-job until the entire computer job is complete. In contrast to attempting to run the computer job as quickly as possible given current resource constraints or to scheduling the job to run "off-hours" to avoid significantly impacting other jobs and applications, the computer job may be run on the computer on an ongoing basis, but in such minute pieces that it may be imperceptible to the user or other computer jobs. Thus, the job may be completely transparent to the user and to other jobs and applications. The user does not need to schedule the job; with this method it can be run at any time, including during performance-critical times.

[0017] In one embodiment, the micro-jobs are inserted for execution from time to time based on selected criteria. In one embodiment, the execution of the micro-jobs is spread out over time based on determining some time interval to space the micro-jobs out by. The time interval does not need to be the same between each of the micro-jobs. In one embodiment, the criterion for micro-job execution is based on resource availability. For example, in order to execute a micro-job, a determination is made as to whether one or more resources used by the micro-job comply with one or more idleness criteria. If the idleness criteria are met, the micro-job is executed.

MICRO-JOBS

[0018] In one embodiment, the micro-jobs have a size that allows a particular micro-job to complete within an allotted time for which the particular micro-job owns a resource used to execute the processing job. In one embodiment, each micro-job is such a size that it will complete within its allotted time. However, it may be that some of the micro-jobs are too large to complete execution within their allotted time.

[0019] In one embodiment, the allotted time is a quantum. As previously discussed, a quantum is a time slice given to a portion of computer code (e.g., a thread) during which time that code portion owns the CPU resource. As also previously discussed, different operating systems used different quanta. Moreover, the quantum assigned to a particular code portion may change based on circumstances during runtime. For example, an O/S might increase or decrease the size of the quantum allotted to a thread. In one embodiment, the computer job is divided into micro-jobs based on the size of the quantum that is expected to be allocated to the computer job. In another embodiment, the computer job is divided into micro-jobs based on the size of the quantum that has been allocated to the computer job. The determination as to what portions of the computer job should be split off as micro-jobs may be made either prior to runtime or during runtime.

[0020] The micro-jobs are substantially smaller (for example, the smallest) work units that can be completed as a single unit while safely allowing for a pause in execution until the next micro-job executes, in accordance with one embodiment. By safely allowing for a pause in execution, it is meant that the execution of a particular micro-job can be delayed without affecting the outcome that results from execution of the all of the micro-jobs.

[0021] A micro-job may be a part of a thread. For example, a thread may be divided into multiple micro-jobs. These micro-jobs may be scheduled similar to how a thread is scheduled. However, as previously stated, a micro-job will complete its execution if allowed to execute for a quantum or other time period for which it owns a processing resource, in one embodiment.

[0022] A micro-job may only need a very small amount of resources (e.g., CPU time, memory allocation) at any one time. Such minimal use of resources at any one time may result in a stealthy process. Keeping the micro-jobs small allows the computer job to use only a small amount of computer resources at one time. Thus, execution of a micro-job consumes a sufficiently small amount of resources so as to not significantly impact performance of other applications in the computer system, in accordance with one embodiment of the present invention.

DIVIDING A COMPUTER JOB INTO MICRO-JOBS

I) APPLICATION DIVIDES ITS COMPUTER JOB INTO MICRO-JOBS

[0023] An application program divides its computer jobs into a plurality of micro-jobs, in one embodiment. As used herein, the term plurality means any number greater than one. FIG. 1 is a diagram of an architecture 100 for executing micro-jobs, in accordance with an embodiment of the present invention. Each MJS-enabled application 115(1) – 115(n) divides its computer job (or jobs) into micro-jobs 125 to execute. For example, an application programmer can place calls at appropriate locations in the application code that request permission from the MJS 110 to execute a micro-job 125, which, in effect, divides the computer job into micro-jobs 125. As examples, the computer job may perform maintenance such as backup, indexing, software updates, virus and malware scans, and defragmentation. However, the MJS-enabled application 115(1) – 115(n) may also be software other than maintenance.

[0024] Still referring to FIG. 1, the micro-job scheduler (MJS) 110 determines when micro-jobs 125 can be executed. Applications that are enabled to work with an MJS are referred to herein as MJS-enabled applications 115. In this embodiment, the MJS 110 has an application program interface (API) 130 to allow a particular MJS-enabled application 115 (e.g., 115(1)) to request that one or more micro-jobs 125 be allowed to execute. The API 130 also allows an MJS-enabled application 115 to specify by how much the micro-jobs 125 may be spread out, as will be discussed more fully below. An example API is included herein below. However, the architecture 100 is not limited to the example API.

[0025] In one embodiment, the MJS 110 maintains a micro-job queue, such that the MJS 110 can determine which micro-job 125 should be allowed to execute next. The execution of micro-jobs 125 is timed by the MJS 110 so as to have negligible impact on other applications.

[0026] The execution of the micro-jobs 125 can be specified by the MJS-enabled application 115 in an API call or other method of communication between the application and the MJS. When the scheduler 105 determines that the next micro-job 125 may execute without impacting other jobs, the MJS 110 responds to the MJS-enabled application 115 by instructing the MJS-enabled application 115(1) to execute the micro-job 125.

[0027] In one embodiment, computer resource utilization is monitored and analyzed to determine whether resource utilization complies with one or more idleness criteria. The MJS causes micro-jobs 125 to be executed when the one or more idleness criteria are satisfied. Since, in one embodiment, the time required for a particular micro-job 125 to execute is less than or equal to a quantum, any resources used by the micro-job 125 are relinquished before the resources are needed by another job. Thus, resource utilization by the micro-job 125 may go unnoticed and the micro-job 125 may be invisible to its application's environment. In one embodiment, the MJS schedules micro-jobs 125 based on time intervals. Time interval based scheduling is discussed below.

[0028] The MJS 110 also has a memory manager 140, in one embodiment. When the MJS 110 initializes, it is allocated memory by the operating system, some of which it uses for its own purposes and some of which it allocates to the MJS-enabled application 115. When an MJS-enabled application 115(1) launches, it requests a memory allocation from the MJS 110. The MJS 110 may determine how much memory to allocate to the MJS-enabled application 115(1) based on factors such as current computer system memory utilization by all processes and the needs of the MJS-enabled application 115(1). Memory requirements may be specific to each MJS-enabled application 115 and may be programmed into the MJS-enabled application 115 by a computer software programmer.

II) SCHEDULER DIVIDES COMPUTER JOB INTO MICRO-JOBS

[0029] In one embodiment, an MJS 110 divides a computer job into micro-jobs 125. Referring to FIG. 2, the MJS 110 has micro-job division logic 210, which divides the computer job 205 into micro-jobs 125. The MJS 110 also have micro-job scheduling logic 220, which schedules the micro-jobs 125 for execution. For example, the MJS 110 may work as a shell wrapped around application programs that are not MJS-enabled. Thus, in this example, the shell MJS 110 is a complete software application through which any executable can be run. Therefore, the shell MJS 110 automatically divides the computer job from the executable into micro-jobs 125, in one embodiment. In other words, the application programmer does not need to divide the application into micro-jobs 125, in this embodiment.

[0030] The shell MJS 110 divides the computer job from the executable into micro-jobs 125 based on resource utilization, in one embodiment. The shell MJS 110 may analyze the application and how the application runs to see what resources the application uses. For example, the shell MJS 110 analyzes what resources the application uses and to what degree the application uses the resources. For example, when a disk defragmenter runs, the shell MJS 110 can determine what resources the applications use (e.g., CPU, network, disk I/O). The shell MJS 110 automatically determines how to divide the application into micro-jobs 125 based on this analysis, in one embodiment. The shell MJS 110 may also determine how to schedule the micro-jobs 125, based on this analysis.

[0031] The shell MJS 110 may use various parameters to determine how to divide the computer job into micro-jobs 125 and/or determine how to schedule the micro-jobs 125 for execution. These parameters may be based on user input or established by the shell MJS 110. For example, the user may specify that a particular application have a high priority.

[0032] The shell MJS 110 schedules the micro-jobs 125 for execution based on time intervals, in one embodiment.

[0033] In one embodiment, the MJS 110 is part of an operating system. In this embodiment, the MJS 110 in the operating system may divide a computer job into micro-jobs 125.

SCHEDULING MICRO-JOBS BASED ON TIME INTERVALS

[0034] In one embodiment, micro-jobs 125 are executed based on time intervals. For example, the MJS 110 schedules micro-jobs 125 based on time intervals. For example, the MJS 110 spreads out execution of the micro-jobs 125 over time. For example, a computer job might take 12 minutes to complete if not divided into micro-jobs 125. However, if divided into micro-jobs 125, the execution of the entire computer job can be spread out over 24 hours, with each particular micro-job 125 executing once every few seconds.

[0035] As a particular example, if the quantum is 20 milliseconds the entire computer job might be completed in about 36,000 quanta. Thus, the computer job is divided into about 36,000 micro-jobs 125. The number of micro-jobs 125 might be slightly higher if some of the micro-jobs 125 are smaller than a quantum. Given, a 20 ms quantum, over a 24-hour period there are 4,320,000 quanta. Therefore, the MJS 110 could execute one of micro-jobs 125 every 120 quanta or every 2.4 seconds. In this particular example, the micro-job enabled application would take 24 hours to complete, but it may be undetectable even on extremely busy systems.

SCHEDULING MICRO-JOBS BASED ON EVENTS

[0036] In one embodiment, micro-jobs 125 are scheduled for execution based on events. For example, the MJS 110 schedules micro-jobs 125 based on a number of operations that occur. As another example, the MJS 110 schedules micro-jobs 125 based on a number of quanta that occur. The different quanta may be of different sizes. Thus, the MJS 110 spreads out execution of the micro-jobs 125 based on events, in one embodiment.

RESOURCE-BASED SCHEDULING OF MICRO-JOBS

[0037] In one embodiment, the micro-jobs 125 are scheduled based on resource utilization. For example, the MJS 110 is a resource-based scheduler, in one embodiment. For example, the MJS 110 schedules micro-jobs 125 such that the micro-jobs 125 utilize only idle resources. The MJS 110 determines whether resource utilization complies with one or more idleness criteria to make scheduling decisions. As an example, the MJS 110 may analyze disk activity. If an application other than the application with the micro-job 125 is using the disk, then the MJS 110 waits until the other application is done to schedule the micro-job 125. The MJS 110 continues to monitor the disk I/O utilization, and allows another micro-job 125 to be scheduled if no other application is seeking access to disk I/O. However, if another application seeks utilization of disk I/O, then the MJS 110 will not allow another micro-job 125 to be scheduled, wherein the other application can utilize the disk I/O.

[0038] As another example, the MJS 110 may analyze network activity. If network traffic is too high, the MJS 110 will not schedule any micro-jobs 125 until traffic slows. If network traffic is low enough, then the MJS 110 schedules a micro job for execution. The MJS 110 continues to check to make sure that network traffic stays low enough. If network traffic stays low enough, another micro-job 125 may be scheduled. However, if traffic gets too high, no further micro-jobs 125 are scheduled to execute.

[0039] The MJS 110 may make resource-based scheduling decisions based on any type of computer resource and any combination of resources. In one embodiment, the MJS 110 has multiple queues of micro-jobs 125 awaiting permission to execute. Each queue may correspond to a particular resource. For example, there may be a queue for micro-jobs 125 that need to utilize disk I/O, a queue for micro-jobs 125 that need to utilize a network, a queue for micro-jobs 125 that need to utilize a CPU, etc. There may also be one or more queues for micro-jobs 125 that utilize a combination of resources. The MJS 110 deploys micro-jobs 125 when the particular resource or combination of resources is available. A particular micro-job 125 might require the use of two resources. For example, the particular micro-job 125 might require use of a network resource and disk resource. However, the particular micro-job 125 does not need CPU resource. Even if the CPU resource utilization is currently high, the particular micro-job 125 can still be scheduled and executed.

[0040] The MJS-enabled application 115 sends parameters to the MJS 110 to control resource utilization, in accordance with an embodiment of the present invention. Control of resource utilization includes, but is not limited to, disk I/O, CPU and network. For example, the MJS-enabled application 115 can request a micro-job 125 be executed pending any combination of threshold levels of the above three resources. Moreover, the MJS-enabled application 115 can specify different resource threshold levels for different micro-jobs 125. For example, the MJS-enabled application 115 specifies a different resource threshold level with each micro-job 125, in accordance with one embodiment. Therefore, fine-grained resource management is possible. When the MJS 110 calculates resource utilization, it is the resource utilization of processes other than the MJS-enabled application 115 that is measured, in accordance with one embodiment of the present invention. The following example in which the CPU utilization threshold is set to 20 percent is used to illustrate. If CPU utilization is below 20 percent prior to allowing the MJS-enabled application 115 to execute, CPU utilization may increase to over 20 percent when the micro-job(s) execute. This increase beyond 20 percent is not considered a CPU resource utilization violation, in this example. Similar principles apply to network and disk I/O resources.

[0041] If the MJS 110 executes outside of the operating system, the MJS 110 self-limits in its own resource utilization, in one embodiment. For example, the MJS 110 monitors its own resource utilization and if its own resource utilization gets too high, the MJS 110 makes a request to the operating system to stop scheduling the MJS 110 for a period of time.

PROCESS FLOW

[0042] FIG. 3 is a flowchart illustrating steps of a process 300 for executing a computer job by dividing the computer job into micro-jobs 125, in accordance with an embodiment of the present invention. In step 302, the computer job is initiated. The computer job may be from an MJS-enabled application 115. However, the computer job does not have to be associated with an MJS-enabled application 115.

[0043] As an example, in step 302, an MJS-enabled application 115 program is launched when the computer system is booted, in one embodiment. If the MJS-enabled application 115 does not have a computer job to perform, the MJS-enabled application 115 sits in an idle state until it has a computer job to run. In this idle state, the MJS-enabled application 115 may perform some functions such as occasional monitoring. At some point, the MJS-enabled application 115 has a computer job to perform, such as defragmenting a storage medium, or scanning for viruses. The computer job could be to defragment a single disk and the files stored on it, wherein the MJS-enabled application 115 defragments that disk on an ongoing basis.

[0044] A small amount of memory may be allocated to the MJS-enabled application 115 when it is launched. The amount that is allocated can be very small because the MJS-enabled application 115 typically only attempts to execute a single micro-job 125 at one time. However, in some cases, the MJS-enabled application 115 might attempt to execute multiple micro-jobs 125 without waiting for other processes to execute. For example, if the MJS 110 determines that the required computer system resources are idle, the MJS 110 may allow the MJS-enabled application 115 to execute multiple micro-jobs 125 in a row without another process utilizing the resources that are utilized by the micro-jobs 125.

[0045] In step 304, the computer job is divided into micro-jobs 125. The size of a particular micro-job 125 allows the particular micro-job 125 to complete within an allotted time for which the particular micro-job 125 owns a resource used to execute the processing job, in one embodiment. The micro-jobs 125 may be of a size such that execution of the micro-jobs 125 utilizes a sufficiently small amount of resources so as to not significantly impact performance of other jobs in the computer system. In one embodiment, dividing the computer job into micro-jobs 125 comprises selecting the size of the micro-jobs 125 such that each micro job 125 is able to complete execution within an allotted time for which the computer job is given ownership of a resource used to execute the micro job 125.

[0046] In one embodiment, the computer job is divided into micro-jobs 125 by an application that owns the computer job. Dividing the computer job into micro-jobs 125 may be accomplished by instructions within an MJS-enabled application 115. In general, these instructions are decision points in the MJS-enabled application 115. For example, the instructions can be API calls to the MJS 110 that request permission to execute a micro-job 125. However, the MJS 110 can be integrated with the MJS-enabled application 115, in which case the instructions could be calls to a scheduling function within the MJS-enabled application 115. Other techniques might be used to divide the computer job into micro-jobs 125. For example, in one embodiment, the computer job is divided into micro-jobs 125 by micro-job division logic in a shell MJS 110.

[0047] Step 306 is execution of the micro-jobs 125. In one embodiment, the entire computer job is divided into micro-jobs 125 and the computer job is then executed micro-job 125 by micro-job 125 until the entire computer job is complete. For example, an entire defragmentation job is divided into micro-jobs 125, which are executed one at a time until the entire defragmentation job is complete. A particular micro job 125 completes its execution in less than or equal to the allotted time, in one embodiment.

[0048] Executing micro-job 125 by micro-job 125 is not limited to sequential one-by-one execution, although it includes sequential one-by-one execution. Multiple micro-jobs 125 could be executed at the same time. For example, if there are multiple CPUs, different micro-jobs 125 could execute on different CPUs at the same time.

[0049] In one embodiment, the scheduling of the micro-jobs 125 is based on resource utilization. In this embodiment, a determination is made, for each of the micro-jobs 125, as to whether utilization of one or more resources of the computer system to be used by the particular micro-job 125 satisfies one or more idleness criteria. The idleness criteria for a particular resource may be based on one or more factors. For example, CPU utilization is used as an idleness criterion for CPU resource utilization, in one embodiment. Thus, the micro-jobs 125 are only executed at times when the resources of the computer system needed by the micro-jobs 125 are sufficiently idle. The idleness criteria are based on resource thresholds, in one embodiment. For example, resource thresholds may be used, wherein a micro-job 125 of an MJS-enabled application 115 only executes if resource utilization by other processes is below a threshold specified by the MJS-enabled application 115. The example API described below contains one example of some resource threshold parameters. However, process 300 is not limited to these resource threshold parameters.

[0050] In one embodiment, the scheduling of the micro-jobs 125 is based on time intervals. In one embodiment, the execution of the computer job is spread over a period of time. For example, the computer job might be spread out over several hours. Based on how many micro-jobs 125 into which the computer job is divided, a determination can be made as to how to spread the micro-jobs 125 out over time. The time between successive micro-jobs 125 does not need to be uniform.

[0051] In one embodiment, an application program that initiated the computer job runs on an on-going basis, continuing to run as long as the computer system remains booted, but remains idle while no work needs to be performed. For example, a disk defragmenter to virus detection application program continues to run. Thus, even if the computer job completes, the application program continues to run in idle mode waiting for the next segment of work. Thus, the application program does not need to be re-launched when it has another computer job to perform. As a result, the application program does not consume additional resources that are typical of launching an application program. When the application program determines that it has another computer job to do, the computer job is divided into micro-jobs 125, and the micro-jobs 125 are executed over time. For example, a disk defragmenter application program may determine that additional storage medium defragmentation should be done based on changes to the computer's storage medium.

EXAMPLE API

[0052] An embodiment of the present invention is an API for allowing an MJS-enabled application 115 to interface with an MJS 110. The example API has the following resource threshold parameters for CPU, disk, and network.

- CPU Utilization threshold
- Pending Disk I/O Count threshold
- Network Utilization threshold

[0053] The above parameters can be specified for each micro-job 125. In other words, different micro-jobs 125 can be assigned different resource threshold parameters. For example, for a micro-job 125 that uses the network, a network threshold may be used. However, the network threshold could be zero for micro-jobs 125 that do not use the network. Thus, fine-grained resource management is provided for, in accordance with an embodiment of the present invention.

[0054] As a particular example, the MJS-enabled application 115 can request that a particular micro-job 125 be executed only if the CPU utilization is below 50%, and the I/O Disk Utilization is below 40%, and network traffic is below 60%. Any combination of the resource threshold factors can be used, including none at all. The CPU utilization threshold differentiates between MJS's use of the CPU as opposed to that of any other job, in accordance with an embodiment of the present invention.

[0055] The following two parameters are used to specify how frequently resource utilization should be measured.

- CPU Utilization Window

- Network Utilization Window

[0056] The CPU Utilization Window parameter defines a time window over which CPU utilization is calculated. For example, CPU utilization over the last *n* milliseconds is averaged. The network utilization window defines a time window over which network utilization is calculated. These parameters may be internal to the MJS 110. However, an MJS-enabled application 115 may override these parameters. The pending disk I/O is absolute at any point in time and thus it does not have to be calculated.

[0057] A mandatory idle time parameter may be passed from the MJS-enabled application 115 to the MJS to control how the micro-jobs 125 are spread out over time. The mandatory idle time parameter is optional. Furthermore, when used, the mandatory idle parameter may have a value of zero.

- Mandatory Idle Time

[0058] The MJS 110 keeps track of "idle time," which is defined as system idle time after all micro-jobs 125 have executed. As previously mentioned, MJS-enabled application 115(s) can queue up micro-jobs 125 with the MJS 110. When there are no micro-jobs 125 on the MJS queue, the MJS 110 waits for the specified Mandatory Idle Time and then wakes up and authorizes the MJS-enabled application 115(s) to perform additional work. For example, an MJS-enabled defragmenter might first execute a number of micro-jobs 125 to defragment a disk drive, and then be paused by the MJS 110. After the specified Mandatory Idle Time, the MJS 110 calls the MJS-enabled defragmenter to authorize additional work. For example, the MJS-enabled defragmenter might execute a clean-up job, such as releasing memory. Mandatory Idle Time can be a default parameter that can be adjusted by an MJS-enabled application 115.

[0059] The following parameters relate to waiting to execute a micro-job 125 when resource utilization is above a threshold.

- Wait Time

- Maximum Wait Time

[0060] If the MJS 110 determines that resource utilization is currently too high to execute a micro-job, the MJS 110 waits for the specified Wait Time and then re-checks resource utilization. The Wait Time parameter can be increased each time the MJS 110 determines that resource utilization is too high. For example, the MJS 110 can increase the Wait Time parameter until the Max Wait Time is reached. These parameters can be specified by the MJS-enabled application 115 when it is first started. An MJS-enabled application 115 can adjust these parameters during its run time.

HARDWARE OVERVIEW

[0061] FIG. 4 is a block diagram that illustrates a computer system 400 upon which an embodiment of the invention may be implemented. Steps of process 300 are stored as instructions one or more of the computer-readable media of system 400 and executed on the processor of computer system 400. Computer system 400 includes a bus 402 or other communication mechanism for communicating information, and a processor 404 coupled with bus 402 for processing information. Computer system 400 also includes a main memory 406, such as a random access memory (RAM) or other dynamic storage device, coupled to bus 402 for storing information and instructions to be executed by processor 404. Main memory 406 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 404. Computer system 400 further includes a read only memory (ROM) 408 or other static storage device coupled to bus 402 for storing static information and instructions for processor 404. A storage device 410, such as a magnetic disk or optical disk, is provided and coupled to bus 402 for storing information and instructions. The computer system 400 can have any number of processors 404. For example, computer system 400 is a multi-processor system, in one embodiment. The processor 404 can have any number of cores. In one embodiment, the processor 404 is a multi-core processor 404. Computer system 400 can be used in a hyper-threaded machine.

[0062] Computer system 400 may be coupled via bus 402 to a display 412, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 414, including alphanumeric and other keys, is coupled to bus 402 for communicating information and command selections to processor 404. Another type of user input device is cursor control 416, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 404 and for controlling cursor movement on display 412. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

[0063] The invention is related to the use of computer system 400 for implementing the techniques described herein. According to one embodiment of the invention, those techniques are performed by computer system 400 in response to processor 404 executing one or more sequences of one or more instructions contained in main memory 406. Such instructions may be read into main memory 406 from another machine-readable medium, such as storage device 410. Execution of the sequences of instructions contained in main memory 406 causes processor 404 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

[0064] The term "machine-readable medium" as used herein refers to any medium that participates in providing data that causes a machine to operate in a specific fashion. In an embodiment implemented using computer system 400, various machine-readable media are involved, for example, in providing instructions to processor 404 for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 410. Volatile media includes dynamic memory, such as main memory 406. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 402. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infrared data communications. All such media must be tangible to enable the instructions carried by the media to be detected by a physical mechanism that reads the instructions into a machine.

[0065] Common forms of machine-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punchcards, papertape, any other physical medium with patterns of holes, a RAM, a PROM, an EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

[0066] Various forms of machine-readable media may be involved in carrying one or more sequences of one or more instructions to processor 404 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 400 can receive the data on the telephone line and use an infrared transmitter to convert the data to an infrared signal. An infrared detector can receive the data carried in the infrared signal and appropriate circuitry can place the data on bus 402. Bus 402 carries the data to main memory 406, from which processor 404 retrieves and executes the instructions. The instructions received by main memory 406 may optionally be stored on storage device 410 either before or after execution by processor 404.

[0067] Computer system 400 also includes a communication interface 418 coupled to bus 402. Communication interface 418 provides a two-way data communication coupling to a network link 420 that is connected to a local network 422. For example, communication interface 418 may be an integrated services digital network (ISDN) card or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 418 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 418 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

[0068] Network link 420 typically provides data communication through one or more networks to other data devices. For example, network link 420 may provide a connection through local network 422 to a host computer 424 or to data equipment operated by an Internet Service Provider (ISP) 426. ISP 426 in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" 428. Local network 422 and Internet 428 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 420 and through communication interface 418, which carry the digital data to and from computer system 400, are exemplary forms of carrier waves transporting the information.

[0069] Computer system 400 can send messages and receive data, including program code, through the network(s), network link 420 and communication interface 418. In the Internet example, a server 430 might transmit a requested code for an application program through Internet 428, ISP 426, local network 422 and communication interface 418.

[0070] The received code may be executed by processor 404 as it is received, and/or stored in storage device 410, or other non-volatile storage for later execution. In this manner, computer system 400 may obtain application code in the form of a carrier wave.

[0071] In the foregoing specification, embodiments of the invention have been described with reference to numerous specific details that may vary from implementation to implementation. Thus, the sole and exclusive indicator of what is the invention, and is intended by the applicants to be the invention, is the set of claims that issue from this application, in the specific form in which such claims issue, including any subsequent correction. Any definitions expressly set forth herein for terms contained in such claims shall govern the meaning of such terms as used in the claims. Hence, no limitation, element, property, feature, advantage or attribute that is not expressly recited in a claim should limit the scope of such claim in any way. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

CLAIMS

What is claimed is:

1. A machine-implemented method comprising the steps:

dividing a computer job into a plurality of micro-jobs, wherein dividing the computer job comprises selecting the size of the micro-jobs based on an allotted time for which the computer job is given ownership of a resource; and

executing the computer job by causing the plurality of micro-jobs to use the resource to execute.

2. The method of Claim 1, wherein selecting the size of the micro-jobs comprises selecting the size of the micro-jobs such that each micro job is able to complete execution within an allotted time for which the computer job is given ownership of a resource used to execute the micro job.

3. The method of Claim 2, wherein the computer job is completed by completing execution of each of the individual micro jobs in less than or equal to the allotted time.

4. The method of Claim 1, wherein executing the computer job comprises:

determining when one or more resources required to run a particular micro-job comply with one or more idleness criteria; and

in response to a determination that utilization of the one or more resources required to run the particular micro-job complies with one or more idleness criteria, causing the particular micro-job to execute to completion within the allotted time.

5. The method of Claim 1, wherein executing the computer job comprises:

determining a length of time to wait between execution of successive micro-jobs.

6. The method of Claim 5, wherein determining the length of time is based on the number of micro-jobs into which the computed job is divided and a period of time over which the computer job is to be spread out over.

7. The method of Claim 1, wherein executing the computer job comprises:

waiting a time interval between execution of successive micro-jobs.

8. The method of Claim 1, wherein executing the computer job comprises:

waiting a number of operations to between execution of successive micro-jobs.

9. The method of Claim 1, wherein executing the computer job comprises:

waiting a number of quanta to between execution of successive micro-jobs.

10. The method of Claim 1, wherein at least two of the quanta are of a different size from each other.

11. The method of Claim 1, wherein at least one of the micro-jobs comprises a portion of a thread.

12. The method of Claim 1, wherein the allotted time is a quantum.

13. A machine-implemented method comprising the steps:

splitting off a micro-job from a computer job that is to execute under control of an operating system that allocates an amount of time for which code portions are given ownership of a resource, wherein the size of the micro job is selected such that the micro job is expected to complete execution within the allotted time; and

causing the micro-job to execute to completion within the allotted time, wherein the micro-job relinquishes ownership of the resource no later than the end of the allotted time.

14. The method of Claim 13, further comprising:

splitting off a plurality of micro-jobs from the computer job, wherein the size of each of the micro jobs is selected such that each micro job is expected to complete execution within the allotted time; and

causing each micro-job to execute to completion within the allotted time for the respective micro job, wherein each micro-job relinquishes ownership of the resource no later than the end of the allotted time for the respective micro job.

15. The method of Claim 13, wherein causing each micro-job to execute causes the computer job to complete.

16. The method of Claim 13, wherein causing the micro-job to execute to completion comprises:

determining when one or more resources required to run the micro-job comply with one or more idleness criteria; and

in response to a determination that utilization of the one or more resources required to run the micro-job complies with one or more idleness criteria, causing the micro-job to execute to completion within the allotted time.

17. The method of Claim 13, wherein the step of splitting off a micro-job comprises:

determining a number of operations of the computer job that can be executed, using the resource, within the allotted time; and

spitting off less than or equal the number of operations from the computer job to form the micro-job.

18. The method of Claim 13, wherein the micro-job comprises a portion of a thread.

19. The method of Claim 13, wherein the allotted time is a quantum.

20. A machine-implemented method comprising the steps of:

dividing, into a plurality of micro-jobs, a computer job that is to execute under control of an operating system that allocates an amount of time for which code portions are given ownership of a resource, wherein dividing the computer job comprises selecting the size of the micro-jobs based on an amount of time for which the computer job is given ownership of a resource; and

spreading execution of the computer job over a period of time.

21. The method of Claim 20, wherein the step of spreading execution of the computer job comprises determining a length of time to wait between execution of two successive micro-jobs of the computer job.

22. The method of Claim 21, wherein determining a length of time is based on a number of micro-jobs into which the computer job is divided and the time period over which the computer job is to be spread out over.

23. The method of Claim 20, wherein the step of spreading execution of the computer job over a period of time comprises:

determining when one or more resources required to run a particular micro-job comply with one or more idleness criteria; and

in response to a determination that utilization of the one or more resources required to run the particular micro-job complies with one or more idleness criteria, causing the particular micro-job to execute to completion within the allotted time.

24. The method of Claim 21, wherein the step of spreading execution of the computer job comprises waiting a length between execution of two successive micro-jobs of the computer job.

25. The method of Claim 21, wherein the step of spreading execution of the computer job comprises waiting a number of operations between execution of two successive micro-jobs of the computer job.

26. The method of Claim 21, wherein the step of spreading execution of the computer job comprises waiting a number of quanta between execution of two successive micro-jobs of the computer job.

27. The method of Claim 21, wherein at least two of the quanta are of a different size from each other.

28. The method of Claim 21, wherein at least one of the micro-jobs comprises a portion of a thread.

29. The method of Claim 21, wherein the allotted time is a quantum.

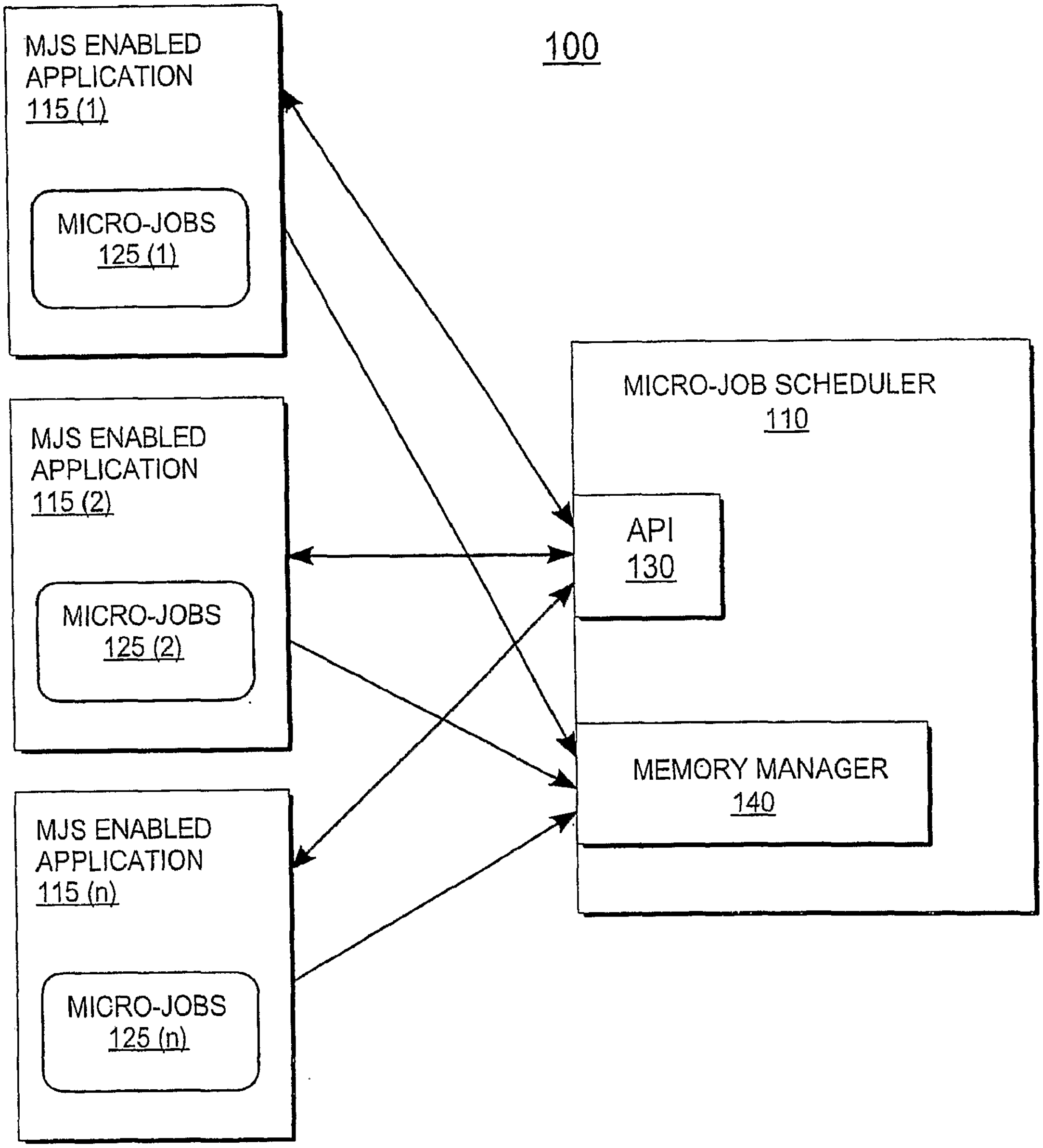

100
MJS ENABLED APPLICATION 115 (1)
MICRO-JOBS 125 (1)
MJS ENABLED APPLICATION 115 (2)
MICRO-JOBS 125 (2)
MJS ENABLED APPLICATION 115 (n)
MICRO-JOBS 125 (n)
MICRO-JOB SCHEDULER 110
API 130
MEMORY MANAGER 140

Fig. 1

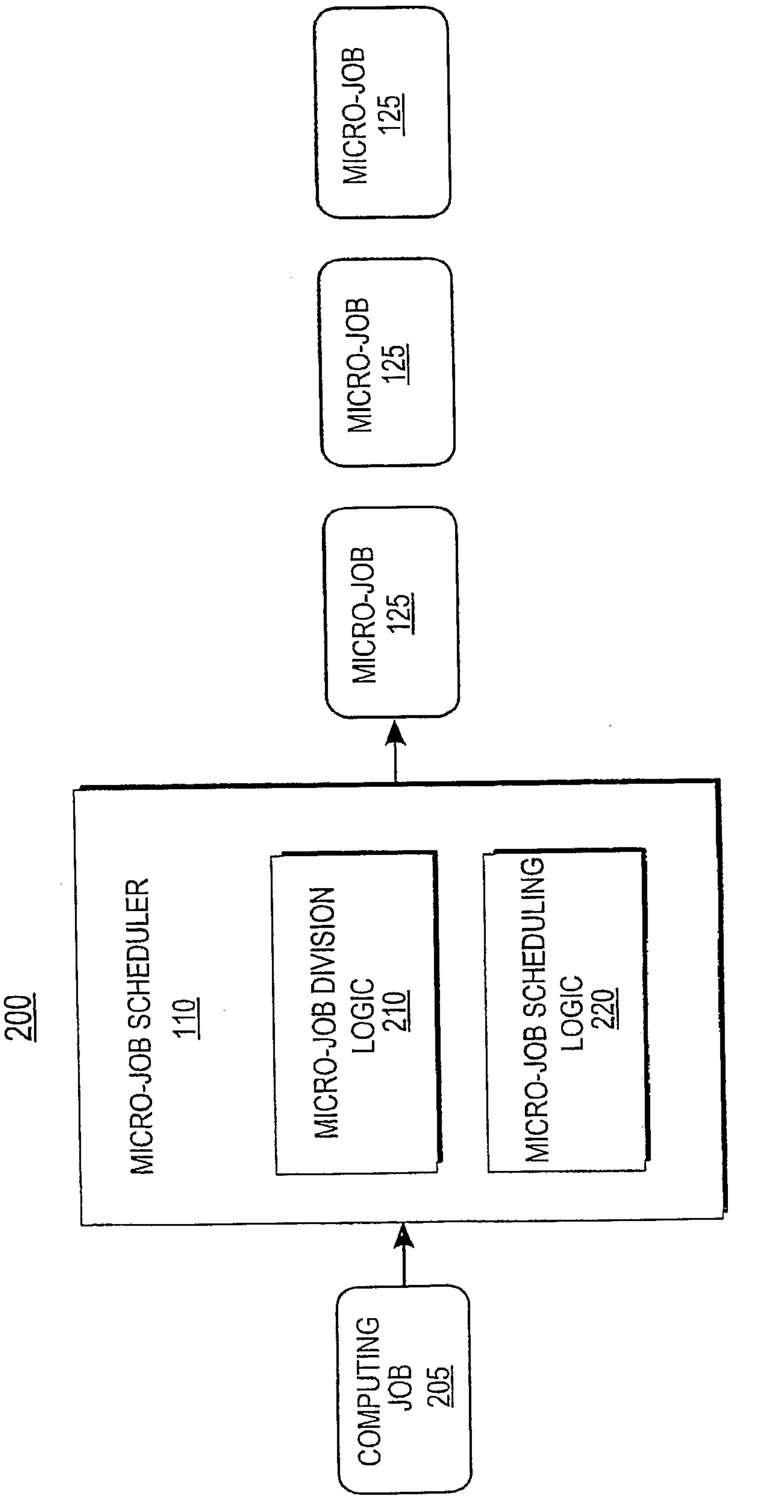

200
COMPUTING JOB 205
MICRO-JOB SCHEDULER 110
MICRO-JOB DIVISION LOGIC 210
MICRO-JOB SCHEDULING LOGIC 220
MICRO-JOB 125
MICRO-JOB 125
MICRO-JOB 125

Fig. 2

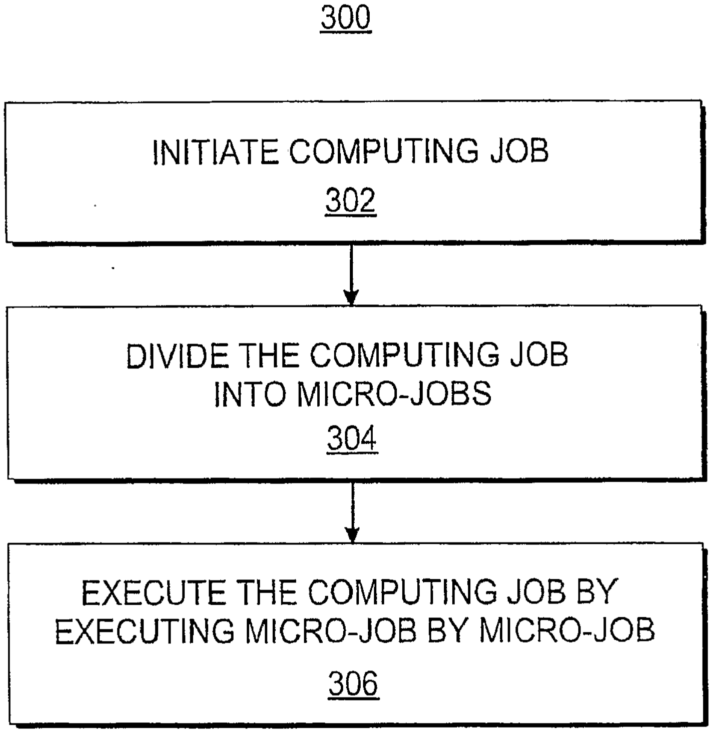

300
INITIATE COMPUTING JOB
302
DIVIDE THE COMPUTING JOB INTO MICRO-JOBS
304
EXECUTE THE COMPUTING JOB BY EXECUTING MICRO-JOB BY MICRO-JOB
306

Fig. 3

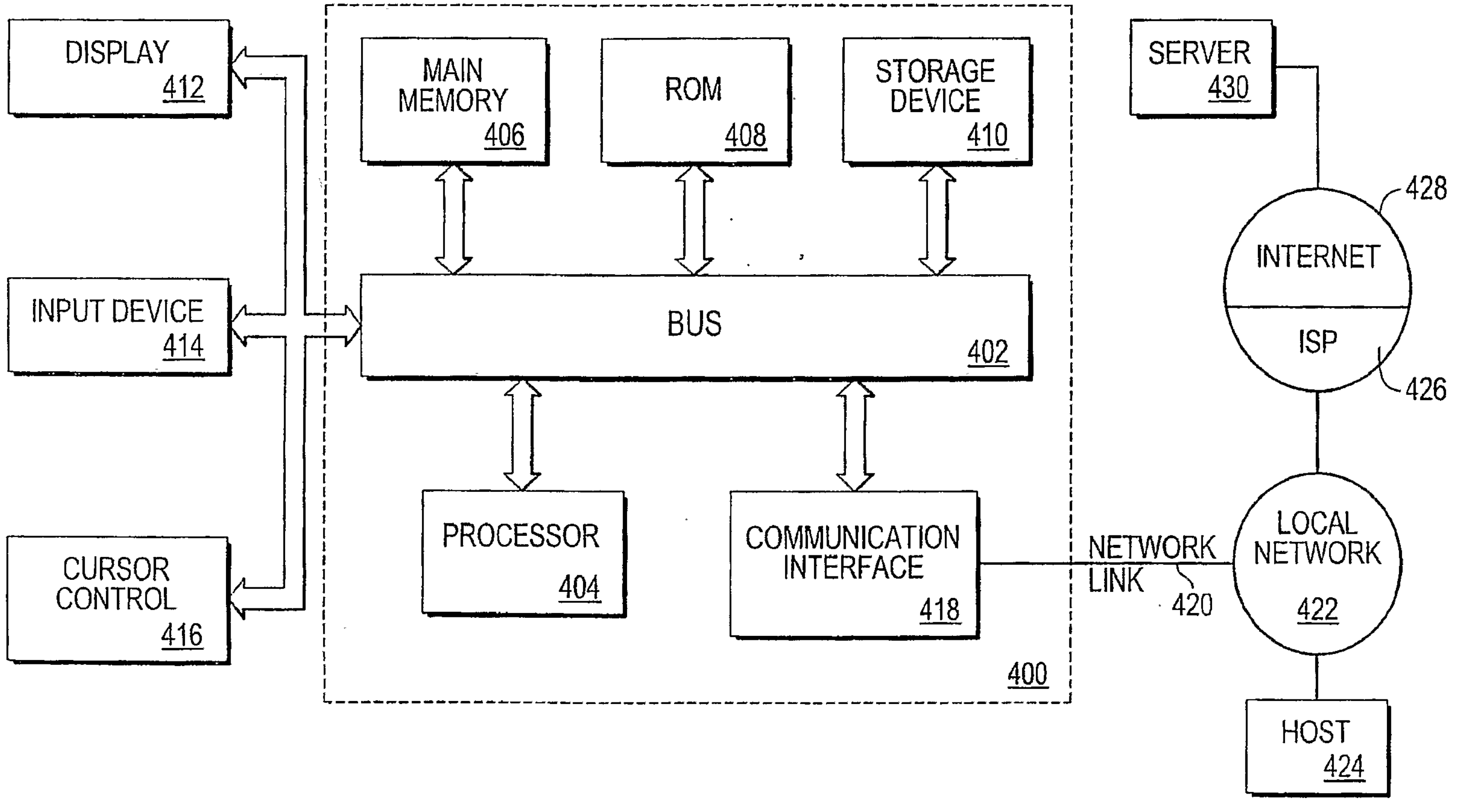

DISPLAY 412
INPUT DEVICE 414
CURSOR CONTROL 416
MAIN MEMORY 406
ROM 408
STORAGE DEVICE 410
BUS 402
PROCESSOR 404
COMMUNICATION INTERFACE 418
400
NETWORK LINK 420
SERVER 430
INTERNET
428
ISP
426
LOCAL NETWORK 422
HOST 424

Fig. 4