PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 11/14, 11/20	A1	(11) International Publication Number: **WO 00/60463** (43) International Publication Date: 12 October 2000 (12.10.00)

(21) International Application Number: PCT/US00/08940

(22) International Filing Date: 5 April 2000 (05.04.00)

(30) Priority Data:
60/127,881 5 April 1999 (05.04.99) US

(71) Applicant: MARATHON TECHNOLOGIES CORPORATION [US/US]; 1300 Massachusetts Avenue, Boxboro, MA 01719 (US).

(72) Inventors: BISSETT, Thomas, D.; 30 Higgins Drive, Kingston, RI 02881 (US). LEVEILLE, Paul, A.; 12 Stratton Road, Grafton, MA 01519 (US). MUENCH, Erik; 677 Salem Street, Groveland, MA 01834 (US). LORD, Christopher, C.; 294 Hubbardston Road, Princeton, MA 01541 (US).

(74) Agents: HAYDEN, John, F. et al.; Fish & Richardson P.C., 601 Thirteenth Street, N.W., Washington, DC 20005 (US).

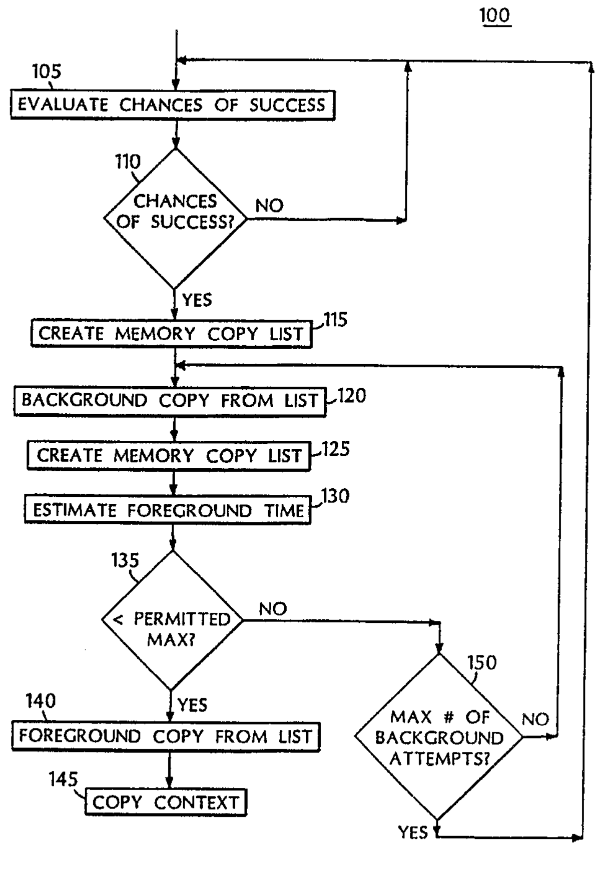
(81) Designated States: AU, CA, JP, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: BACKGROUND SYNCHRONIZATION FOR FAULT-TOLERANT SYSTEMS

(57) Abstract

An inactive memory is synchronized with an active memory in a fault-tolerant computer system that includes an active processor. Data is copied from the active memory to the inactive memory using a background process that permits the active processor to perform normal operations while the copying is proceeding. Regions of the active memory in which changes are made are tracked while the copying is proceeding, and, after copying is complete, a determination is made as to whether data from the regions of the active memory in which changes were made can be copied to the inactive memory within a predetermined time period using a foreground process that prevents the active processor from performing normal operations. If the data can be copied to the inactive memory within the predetermined time period using the foreground process, the data is copied to the inactive memory using the foreground process. If the data cannot be copied to the inactive memory within the predetermined time period using the foreground process, the copying, tracking, and determining are repeated for the regions of the active memory in which changes were made.



BACKGROUND SYNCHRONIZATION FOR FAULT-TOLERANT SYSTEMS

TECHNICAL FIELD

The invention relates to restoring synchronized execution by processors in fault resilient/fault tolerant computer systems.

BACKGROUND

Computer systems that are capable of surviving hardware failures or other faults generally fall into three categories: fault resilient, fault tolerant, and disaster tolerant.

Fault resilient computer systems can continue to function, often in a reduced capacity, in the presence of hardware failures. These systems operate in either an availability mode or an integrity mode, but not both. A system is "available" when a hardware failure does not cause unacceptable delays in user access, which means that a system operating in an availability mode is configured to remain online, if possible, when faced with a hardware error. A system has data integrity when a hardware failure causes no data loss or corruption, which means that a system operating in an integrity mode is configured to avoid data loss or corruption, even if the system must go offline to do so.

Fault tolerant systems stress both availability and integrity. A fault tolerant system remains available and retains data integrity when faced with a single hardware failure, and, under some circumstances, when faced with multiple hardware failures.

Disaster tolerant systems go beyond fault tolerant systems. In general, disaster tolerant systems require that loss of a computing site due to a natural or man-made disaster will not interrupt system availability or corrupt or lose data.

All three cases require an alternative component that continues to function in the presence of the failure of a component. Thus, redundancy of components is a fundamental prerequisite for a disaster tolerant, fault tolerant or fault resilient system that recovers from or masks failures. Redundancy can be provided through passive redundancy or active redundancy, each of which has different consequences.

A passively redundant system, such as a checkpoint-restart system, provides access to alternative components that are not associated with the current task and must be either activated or modified in some way to account for a failed component. The consequent transition may cause a significant interruption of service. Subsequent system performance also may be degraded. Examples of passively redundant systems include stand-by servers and clustered systems. The mechanism for handling a failure in a passively redundant system is to "fail-over," or switch control, to an alternative server. The current state of the failed application may be lost, and the application may need to be restarted in the other system. The fail-over and restart processes may cause some interruption or delay in service to the users. Despite any such delay, passively redundant systems such as stand-by servers and clusters provide "high availability" and do not deliver the continuous processing usually associated with "fault tolerance."

An actively redundant system, such as a replication system, provides an alternative processor that concurrently processes the same task and, in the presence of a failure, provides continuous service. The mechanism for handling failures is to compute through a failure on the remaining processor. Because at least two processors are looking at and manipulating the same data at the same time, the failure of any single component should be invisible both to the application and to the user.

The goal of a fault tolerant system is to produce correct results in a repeatable fashion. Repeatability ensures that operations may be resumed after a fault is detected. In a checkpoint-restart system, this entails rolling back to a previous checkpoint and replaying the inputs again from a journal file. In a replication system, repeatability results from simultaneous operation on multiple instances of a computer.

Processes performed when a fault occurs in an actively redundant system may include fault detection, fault isolation, fault recovery, repair, and system restoration (including synchronization). For example, when application instructions are executed in the same order on all copies of a processor, and a fault occurs in one copy of the processor:

1. The fault is detected.
2. The fault is identified as coming from a particular copy of the processor and effects of the fault are constrained so as not to adversely affect the system.
3. The system recovers from the fault and continues with no side effects to the application, but with a reduced level of fault tolerance.
4. The faulty processor is repaired or replaced.

5. The repaired processor synchronizes itself with the remaining processors to restore the system's normal level of fault tolerance.

The synchronization process may be performed as a foreground process or a background process. Foreground synchronization takes complete control of the processors and dedicates them to copying all memory contents and processor context information to the synching processor. Background synchronization copies the memory contents to the synching processor as a background process (i.e., while application programs continue to run), and then switches to a short foreground synchronization process to copy processor context information.

Foreground synchronization consumes 100% of the processors' operating cycles for the duration of the memory copy. This locks out any application programs being run by the processors from all external devices, which may be a problem if the memory copy takes too long. For example, with some network protocols, a network connection that is not serviced at least once every six seconds can be dropped. This places a restriction on the maximum memory size that can be supported without making special provisions in the application program for the fault tolerant state of the system, which is undesirable.

The maximum memory size that can be supported for a particular maximum synchronization time can be increased by increasing the I/O bandwidth in a direct, linear relationship with the increase in memory size. In recent years, desired growth in memory size has outpaced the I/O bandwidth of computers, making it difficult to synchronize desired memory sizes with foreground synchronization.

Background synchronization renders the linear relationship between I/O bandwidth and memory size unnecessary by allowing the memory copy to occur while the application is still running. The timing constraint associated with background synchronization is that the duration of the foreground process at the end of the bulk memory copy must be less than the permitted maximum (e.g., six seconds).

One prior approach to background synchronization was to sweep all memory with direct processor read/writes or with direct memory access ("DMA"). Every location touched was transferred to the synching processor. This sweep was done while application processes were running. Any location that an application process modified also needed to be transferred. This was achieved using a custom memory controller that, every time that a memory write occurred, automatically transferred the address/data pair associated with the memory write to the synching processor. This approach guaranteed that, at the end of the memory sweep, all memory had been transferred. The final foreground task consisted of sending the processor context to the synching processor.

The background sweep could be stretched out over seconds to hours depending on the memory size and the system I/O bandwidth that the operator was willing to dedicate to the synchronization process. The final foreground part of the synchronization occurred in less than one second.

SUMMARY

In one general aspect, the invention features synchronizing an inactive memory with an active memory in a fault-tolerant computer system that includes an active processor. Data is copied from the active memory to the inactive memory using a background process that permits the active processor to perform normal operations while the copying is proceeding. Regions of the active memory in which changes are made are tracked while the copying is proceeding, and, after copying is complete, a determination is made as to whether data from the regions of the active memory in which changes were made can be copied to the inactive memory within a predetermined time period using a foreground process that prevents the active processor from performing normal operations. If the data can be copied to the inactive memory within the predetermined time period using the foreground process, the data is copied to the inactive memory using the foreground process. If the data cannot be copied to the inactive memory within the predetermined time period using the foreground process, the copying, tracking, and determining are repeated for the regions of the active memory in which changes were made.

Implementations may include one or more of the following features. For example, an evaluation may be made as to whether the synchronizing is likely to be successful prior to copying data from the active memory to the inactive memory using the background process. Such an evaluation may be accomplished in one of several ways. For example, evaluating whether the synchronizing is likely to be successful may include comparing a rate at which data in the active memory are modified to a rate at which data can be transferred from the active memory to the inactive memory using the background process.

When the result of the evaluating indicates that the synchronizing is not likely to be successful, efforts may be made to mitigate the problem. Mitigation may include, for example, increasing an amount of bandwidth allocated to the background process prior to repeating the copying, tracking, and determining for the regions of the active memory in which changes were made. Mitigation also may include restricting an amount of working memory for one or more running applications to a minimum amount that still permits the one or more running applications to run prior to repeating the copying, tracking, and determining for the regions of the active memory in which changes were made. In addition, a data compression operation may be performed on the data from the regions of active memory in which changes were made prior to repeating the copying, tracking, and determining for the regions of the active memory in which changes were made.

Active memory may be associated with the active processor. The active processor may include a compute element and an I/O processor, where the compute element implements the copying, tracking and determining.

A memory copy list identifying portions of the active memory for which data are to be copied to the inactive memory using the background process may be created and used in copying data using the background process. Tracking regions of the active memory may include creating a new memory copy list.

When the computer system includes an inactive processor associated with the inactive memory, and the active processor is associated with the active memory, the context of the active processor may be copied to the inactive processor.

Tracking regions of the active memory may include using a page table structure including pages of memory and corresponding page table entries, with each page table entry including an indicator bit that is set when a memory location of the corresponding page of memory is modified. In another example, the active processor includes a memory control section and tracking regions of the active memory includes using a memory block structure allocated by the memory control section, including updating a flag corresponding to a block of memory whenever the block of memory is modified.

When the data from the regions of the active memory in which changes were made cannot be copied to the inactive memory within the predetermined time period using the foreground process, the method may include, for example, increasing an amount of bandwidth allocated to the background process prior to repeating the copying, tracking, and determining for the regions of the active memory in which changes were made. As an alternative, or in addition, an amount of working memory for one or more running applications may be restricted to a minimum amount that still permits the one or more running applications to run prior to repeating the copying, tracking, and determining. Also, a data compression operation may be performed on the data from the regions of active memory in which changes were made prior to repeating the copying, tracking, and determining. These techniques also may be used prior to the initial copying of data from the active memory to the inactive memory using the background process.

A number of pages of active memory may be allocated to the synchronization process prior to copying data from the active memory to the inactive memory using the background process. Prior to the allocation, the inactive memory is cleared, and only unallocated pages are copied.

Another implementation may include clearing the inactive memory and determining which regions of the active memory contain nonzero data. In this implementation, copying data from the active memory to the inactive memory using the background process includes copying data only from the regions of the active memory that contain nonzero data. The active processor may include a memory control section, and determining which regions of the active memory contain nonzero data then may include using the memory control section to store a list of which regions of active memory contain nonzero data.

The details of one or more embodiments of the invention are set forth in the accompanying drawings and the description below. Other features, objects, and advantages of the invention will be apparent from the description and drawings, and from the claims.

DESCRIPTION OF DRAWINGS

Fig. 1 is a flow chart that illustrates a procedure for providing background synchronization for a fault tolerant system.

Figs. 2 and 3 are block diagrams of a fault tolerant system that emulates clock lockstep operation.

Fig. 4 is a flow chart that illustrates a procedure for providing background synchronization for a fault tolerant system using a software-only, page table tracking implementation.

Fig. 5 is a flow chart that illustrates a procedure for providing background synchronization for a fault tolerant system using a software-only, balloon zeroing implementation.

Fig. 6 is a flow chart that illustrates a procedure for providing background synchronization for a fault tolerant system using a hardware-assisted memory controller tracking method.

Like reference symbols in the various drawings indicate like elements.

DETAILED DESCRIPTION

A heuristic approach may be used to provide background synchronization without requiring a custom memory controller. The heuristic approach attempts to perform the foreground part of the synchronization in less than the permitted maximum synchronization time (e.g., six seconds). The background task is not limited in time or in the number of memory sweeps performed.

Referring to Fig. 1, the heuristic approach operates according to a procedure 100. First, the chances that the synchronization will complete successfully are evaluated (step 105). As discussed in more detail below, this evaluation may involve a comparison of the rate at which memory is copied to the rate at which memory is modified. If there is no chance of success (step 110), the evaluation is repeated (step 105), with the hope that system conditions will have changed in a way that permits synchronization to complete successfully. The evaluation may be repeated upon expiration of a fixed delay period.

If there is some chance of success, a memory copy list is created (step 115). Next, all blocks of memory in the memory copy list are copied to the synching processor using a background process (step 120).

A new memory copy list then is created (step 125). This new list identifies all memory blocks that have been modified since the last memory copy list was created. From the new memory copy list, the time required to perform a foreground synchronization process is estimated (step 130).

If the estimated time for foreground synchronization is less than the permitted maximum (e.g., six seconds) (step 135), then the memory copy is completed using foreground synchronization (step 140). After memory copy is completed, the processor context is copied to the synching processor in the foreground to complete the synchronization procedure (step 145).

If the estimated time for foreground synchronization is greater than the permitted maximum (step 135), and background memory copy has not been attempted a maximum number of times (step 150), all blocks of memory in the memory copy list are copied to the synching processor using a background process (step 120). The procedure then proceeds as described above.

If the estimated time for foreground synchronization is greater than the permitted maximum (step 135), and background memory copy has been attempted a maximum number of times (step 150), this attempt at synchronization has failed. Upon failure of synchronization, the chances that the synchronization will complete successfully are again evaluated (step 105).

A memory block structure is used to track the areas of memory that have been modified and to create the memory copy lists. The memory block structure may be provided in several different ways.

The most basic way is a page table structure. This structure is provided by processors currently available from Intel Corporation. Each page table entry ("PTE") includes a dirty bit that is set when a memory location of the page is modified. The dirty bit is set by hardware, and is not altered by the operating system. The PTEs can be used to track which pages of memory are modified while the background memory copy is performed.

The PTEs provide a very detailed list of modified pages. For example, for a one gigabyte memory, there are up to one million page table entries. This list may be too detailed to use for synchronization purposes.

An alternate approach is to add a tracking mechanism to the memory control section of the motherboard chip set. A section of memory (part of system memory, on-chip memory, or dedicated external memory) can be allocated to a modified block structure. Whenever a block of memory is written, a corresponding flag in the modified block structure is updated. Between passes of the background copy process, a snapshot of the modified block structure is made to control the next pass of the background copy process.

There is a tradeoff between the resolution of the modified block structure and the transfer time for a block. Each background copy pass requires scanning the structure and setting up transfers (i.e., creating the memory copy list). The transfer time for each block is directly related to the block size, while the scan time is inversely related to the block size.

To evaluate whether the synchronization will complete successfully (step 105), the block modification rate is monitored to see if a background synchronization procedure will ever finish. For example, if two blocks get modified during the time required to copy one block, then the background synchronization will never finish.

To create the first memory copy list (step 115), a list of all blocks of memory is created. To create subsequent memory copy lists (step 125), a list of all modified blocks of memory is created. In either case, the PTEs or the modified block structure are cleared after the memory copy list is created.

The procedure is not guaranteed to complete. If the block modification rate is too high, there will always be more modified memory blocks to copy than can be copied in the permitted foreground copy time. Several refinements can be made to improve the chances of success. In a first example, the bandwidth that the background copy task is allowed to consume is increased. In a second example, any running applications are restricted down to their minimum working memory sets. In a third example, rudimentary data compression is performed on the memory images before transfer. Particular implementations may use one or more of these techniques.

The more bandwidth that is allowed to the background synchronization copy, the fewer blocks of memory the applications will have time to modify during the background copy. At 100% background allocation, synchronization will always succeed. However, this is likely to violate the maximum synchronization time requirement, since background synchronization with a 100% allocation is really foreground synchronization.

The synchronization process can allocate large chunks of system memory to itself. This memory then can be zeroed in both processors, so that it does not need to be copied. In addition, synchronization allocation restricts the number of memory pages available for applications to modify.

To implement synchronization allocation, the synchronization process starts with a small set of pages and expands the number of pages until the operating system prevents the process from taking any more. The synchronization process runs as a driver under the operating system, which means that the process is allowed to lock down pages of memory so that they are never sent out to disk storage. Accordingly, any pages given to the synchronization process represent pages of physical memory that do not have to be copied. The physical memory available to applications that are running is compressed until the applications are close to their minimum working sets.

The memory data to be copied may be compressed before a copy. A benefit of such a compression is that the processor-to-memory bandwidth is much greater than the memory-to-I/O bandwidth. As a result, a simple compression scheme can reduce the total copy time significantly.

Referring to Fig. 2, the procedure 100 may be implemented by a fault tolerant system 200, such as the Endurance® 4000 system, which is available from Marathon Technologies Corporation of Boxboro, Massachusetts. The system operates multiple instances of a compute element in instruction stream lockstep, which means that the multiple instances of a compute element perform the same sequence of instructions in the same order. By contrast, fully phase-locked operation, which also may be referred to as clock lockstep operation, occurs when multiple instances of a compute element perform the same sequence of instructions in the same order, with each instruction being performed in the same clock cycle by each instance of the compute element.

As noted, the instances of a compute element in the system 200 operate in instruction stream lockstep. The time needed to execute the instruction stream varies due to the uncontrolled past history of each compute element. For example, caches, table look-ahead buffers, branch prediction logic, speculative execution logic, and execution pipelines of the compute elements can have different initial values, which, even though the instruction streams being executed are the same, result in varying execution times.

Clock lockstep operation may be achieved by using a common oscillator to provide clocks to all instances of the compute element. However, such an implementation may be unsuited for fault tolerant operation because it includes a single component, the common oscillator, the failure of which will cause failure of the entire system.

Emulated clock lockstep operation avoids the single point of failure and is achieved using the techniques described below. Emulated clock lockstep operation offers the considerable additional benefit of permitting the different instances of a compute element to be separated by distances of up to a kilometer or more.

In general, all computer systems perform two basic operations: (1) manipulating and transforming data, and (2) moving the data to and from mass storage, networks, and other I/O devices. The system 200 divides these functions, both logically and physically, between two separate processors. For this purpose, each half of the system 200, called a tuple, includes a compute element ("CE") 205 and an I/O processor ("IOP") 210. The compute element 205 processes user application and operating system software. Thus, the compute element 205 implements the procedure 100. I/O requests generated by the compute element 205 are redirected to the I/O processor 210. This redirection is implemented at the device driver level. The I/O processor 210 provides I/O resources, including I/O processing, data storage, and network connectivity. The I/O processor 210 also controls synchronization of the compute elements.

The system 200 is fault tolerant in that it continues to operate transparently to its users in the presence of any single hardware failure. The system 200 emulates a traditional computing environment by partitioning the environment into two components. The compute element 205 handles all computing tasks for the operating system and any applications. The I/O processor 210 handles all I/O devices. Thus, the I/O processor 210 handles all of the asynchronous activities associated with a computer, while the compute element 205 handles all of the synchronous computing activities.

To provide the necessary redundancy for fault tolerance, the system 200 includes at least two compute elements 205 and at least two I/O processors 210. The two compute elements 205 operate in lockstep while the two I/O processors 210 are loosely coupled. The I/O processors 210 feed both compute elements 205 the exact same data at a controlled place in the instruction streams of the compute elements. The I/O processors 210 verify that the compute elements 205 generate the same I/O operations and produce the same output data at the same time. The I/O processors 210 also cross check each other for proper completion of requested I/O activity.

The system 200 uses a software-based approach in a configuration based on inexpensive, industry standard processors. For example, the compute elements 205 and I/O processors 210 may be implemented using Pentium Pro processors available from Intel Corporation. The system may run unmodified, industry-standard operating system software, such as the Windows NT operating system available from Microsoft Corporation, as well as industry-standard applications software. This permits a fault tolerant system to be configured by combining off-the-shelf, Intel Pentium Pro-based servers from a variety of manufacturers, which results in a fault tolerant or disaster tolerant system with low acquisition and life cycle costs.

Each compute element 205 includes a processor 215, memory 220, and an interface card 225 (also referred to as a Marathon interface card, or MIC). The interface card 225 includes drivers for communicating with two I/O processors simultaneously, as well as comparison and test logic that assures results received from the two I/O processors are identical. In the fault tolerant system 200, the interface card 225 of each compute element 205 is connected by high speed links 230, such as fiber optic links, to interface cards 225 of the two I/O processors 210. The interface cards 225 may be implemented as PCI-based adapters.

Each I/O processor 210 includes a processor 215, memory 220, an interface card 225, and I/O adapters 235 for connection to I/O devices such as a hard drive 240 and a network 245. As noted above, the interface card 225 of each I/O processor 210 is connected by high speed links 230 to the interface cards 225 of the two compute elements 205. In addition, a high speed link 250, such as a private Ethernet link, is provided between the two I/O processors 210.

All I/O task requests from the compute elements 205 are redirected to the I/O processors 210 for handling. The I/O processor 210 runs specialized software that handles all of the fault handling, disk mirroring, system management, and resynchronization tasks required by the system 200. By using a multitasking operating system, such as Windows NT, the I/O processor 210 may run other, non-fault tolerant applications. In general, a compute element may run Windows NT Server as an operating system while, depending on the way that the I/O processor is to be used, an I/O processor may run either Windows NT Server or Windows NT Workstation as an operating system.

The two compute elements 205 run lockstep control software, also referred to as quantum synchronization software, and execute the operating system and the applications in emulated clock lockstep. Disk mirroring takes place by duplicating writes on the disks 240 associated with each I/O processor 210. If one of the compute elements 205 should fail, the other compute element 205 keeps the system running with a pause of only a few milliseconds to remove the failed compute element 205 from the configuration. The failed compute element 205 then can be physically removed, repaired, reconnected, and turned on. The repaired compute element then is brought back automatically into the configuration by transferring the state of the running compute element to the repaired compute element over the high speed links 230 and resynchronizing using, for example, the procedure 100. The states of the operating system and applications are maintained through the few seconds it takes to resynchronize the two compute elements 205 so as to minimize any impact on system users.

If an I/O processor 210 fails, the other I/O processor 210 continues to keep the system running. The failed I/O processor then can be physically removed, repaired and activated. Since the I/O processors are not running in lockstep, the repaired system may go through a full operating system reboot, and then may be resynchronized. After being resynchronized, the repaired I/O processor automatically rejoins the configuration and the mirrored disks are re-mirrored in background mode over the private connection 250 between the I/O processors 210. A failure of one of the mirrored disks is handled through the same process.

The connections to the network 245 also are fully redundant. Network connections from each I/O processor 210 are booted with the same address. Only one network connection is allowed to transmit messages, while both are allowed to receive messages. In this way, each network connection monitors the other through the private Ethernet 250. Should either network connection fail, the I/O processors 210 will detect the failure and the remaining connection will carry the load. The I/O processors 210 notify the system manager in the event of a failure so that a repair can be initiated.

While Fig. 2 shows both connections on a single network segment, this is not a requirement. Each I/O processor's network connection may be on a different segment of the same network. The system also accommodates multiple networks, each with its own redundant connections. The extension of the system to disaster tolerance requires only that the connection between the tuples be optical fiber or a connection having compatible speed. With such connections, the tuples may be spaced by distances of a kilometer or more. Since the compute elements are synchronized over this distance, the failure of a component or a site will be transparent to the users.

Fig. 3 provides a summarized view of the system 200 of Fig. 2. The system includes redundant compute elements 205 ("CEs") and I/O processors 210 ("IOPs"). Each CE 205 is responsible for all computing and may be implemented using an industry standard motherboard. Each IOP 210 is responsible for access to I/O devices, and for system control. The IOPs 210 run asynchronously of each other and verify that the CEs 205 are performing the same operations in the same order. The IOPs 210 also track each other's I/O completion to ensure that no I/O is lost.

The CEs 205 generate the same outputs in the exact same sequence, and run in emulated clock lockstep, even though the CE clocks are asynchronous to each other. The CEs 205 are initialized to the same state and are fed consistent inputs at exactly the same time. The CEs 205 are periodically realigned using a self-generated interrupt that is related to the occurrence of a quantum of clock cycles (e.g., 100,000 clock cycles) and is referred to as a quantum interrupt ("QI"). All inputs to the CEs 205 are delivered at either an output window or after the completion of an instruction quantum. Both of these points are guaranteed to occur at the same point in the instruction streams of the CEs 205. The approach employed by the system 200 is described in U.S. Patent Nos. 5,600,784 and 5,615,403, both of which are incorporated by reference.

Foreground Synchronization

A CE must be synchronized back into the system 200 following removal of the CE. The CE may have been removed for any number of reasons: a transient failure, a hard failure and repair, or even a scheduled removal. To rejoin the system, a foreground synchronization procedure may be used to transfer the static state of a suspended CE to a synchronizing CE. A large part of this procedure involves the transfer of CE main memory.

The CE operating system is suspended for the duration of the foreground synchronization procedure. This suspension is visible to users, since applications and network communications are temporarily stopped. The CE operating system resumes operation after the synchronization procedure has completed. The temporary suspension, however, may cause network session timeouts or exceed a user's requirements for application dead time.

Network connections are able to survive a full foreground synchronization on systems that adhere to the 128 MB guideline for CE memory capacity. At 16 MB/s, foreground synchronization of 128 MB is typically completed in approximately eight seconds. Users with more than this amount of memory are advised to disable automatic CE synchronization and, as necessary, to initiate the synchronization procedure at a convenient time of day or night. Although this may be a viable work-around for some users, it is unacceptable to others.

Some users wish to impose a rigid limit on the amount of time an application can be suspended, for any reason. Foreground synchronization will typically exceed this limit, requiring the user to disable automatic CE synchronization.

One of the major benefits of the system 200 is its hands-off operation. Components are automatically removed, joined (IOPs), mirrored, and synchronized as necessary to maintain a high level of availability. This benefit cannot be fully realized by users that need to run with larger CE memory sizes, yet have connectivity or real-time-like constraints. These users will need to disable automatic CE synchronization and attempt to find a safe time to manually initiate a resynchronization.

Permitted memory sizes may be increased by increasing the speed of the CE-to-CE interconnect (i.e., the MIC). However, as shown below in Table 1, modest speed improvements alone are not likely to reduce the foreground synchronization time to acceptable levels. More aggressive interconnect speeds are possible, but only at much higher cost or by imposing distance restrictions. The foreground synchronization rates of 50 MB/s and 100 MB/s are used here for illustrative purposes only.

CE Memory Size	**Foreground Synch @ 50 MB/s**	**Foreground Synch @ 100 MB/s**	**Ideal CE Synch Time**
1 GB	20 seconds	10 seconds	< 6 seconds
2 GB	40 seconds	20 seconds	< 6 seconds
3 GB	60 seconds	30 seconds	< 6 seconds
4 GB	80 seconds	40 seconds	< 6 seconds
8 GB	160 seconds	80 seconds	< 6 seconds

Table 1 - Foreground Synchronization Times

The ideal CE synchronization time is based on the worst-case session timeout period for a protocol, such as TCP/IP, used by the system. In general, such a protocol will sustain connections over longer periods of silence, but the exact time tolerated is determined and adjusted dynamically by the protocol stacks. Tighter limits may be imposed by users with real-time or substantially-real-time application requirements.

Background Synchronization

As discussed above, background synchronization refers to the process of transferring portions of a running CE's memory context to a synchronizing CE, without suspension of the operating system. The CE operating system and applications are unaware of this controlled-rate transfer, although the transfer does consume some portion of the MIC's available bandwidth. As the transfer is taking place, the CE operating system continues to run applications and service network clients, with some tolerable level of degraded performance. After many seconds of background transfer, depending on CE memory size and other parameters, the CE operating system is suspended and foreground synchronization is performed to transfer areas modified during the background synchronization.

The CE operating system workload profile can outpace the background memory transfer such that the ensuing foreground synchronization will not complete within the desired target time interval. However, software can pre-determine the foreground synchronization time and choose to abort the synchronization process if user-selected limits are exceeded. This allows automatic CE synchronization features to remain enabled, ensuring that network and application timeout limits will not be exceeded.

The goals of the background synchronization are to allow all users to run the system with automatic CE synchronization enabled, and to ensure that foreground synchronization will never exceed a time limit established by the user. With these goals in mind, it is clear that the overall time required to integrate a CE into the system is not particularly important. Some degradation of CE operating system performance is also permissible during synchronization, and this will likely also be settable by the user.

Software-Only, Page Table Tracking Implementation

Referring to Fig. 4, this approach to a software-only implementation of background synchronization employs the Pentium® architecture's 'dirty' indicators. These indicators are provided for each page table entry and can be used to track processor modifications to memory. The procedure 400 includes the same steps as the procedure 100 of Fig. 1, and adds two additional steps. First, prior to performing a background copy of all pages to the target CE (i.e., after step 115 and before step 120), various page-table maintained indicators are set or cleared (step 405). Following the background copy (step 120), these indicators are rechecked to determine what pages were modified during the copy (step 410), and, therefore, must be copied again during a subsequent background copy or during a foreground completion phase.

Advantages to this approach are that it works on existing Pentium-based systems, it is not tied to hardware chip schedules, it permits page-level granularity of memory modifications, and it should provide convergence on an acceptable foreground synchronization time under most loads. Potential drawbacks of this approach are that its implementation is operating system-specific, and that page tables only track processor-originated memory modifications, and do not track adapter direct memory access ("DMA"). This approach also is sensitive to operating system process context switching (e.g., page directory, and PTE management), and may require access to the operating system source code to understand context-switch issues. Finally, this approach raises the risk that unforeseen operating system behaviors, and future operating system changes, may cause problems.

Software-Only, Balloon Zeroing Implementation

Referring to Fig. 5, a balloon zeroing procedure 500 provides a simpler approach to a software-only implementation. Instead of performing a background copy of all memory, this implementation begins by having the target CE clear all of its physical memory (step 505). The master CE then drops into a foreground synchronization mode and transfers only pages that contain non-zero data (step 510). This approach capitalizes on the observation that zero-filled pages are quite common in most virtual-memory operating systems. In addition, cooperative threads can be used to forcibly zero out a large portion of memory just prior to attempting the foreground synchronization. This pre-zeroing effort is referred to as balloon zeroing.

Advantages of this approach are that it works on existing Pentium-based systems; it is not tied to hardware chip schedules; it is very simple to implement; and it permits page-level granularity of zero/non-zero data. Potential disadvantages of this approach are that the scan memory time to determine if foreground synchronization should be performed now may take much more time, depending on memory size and processor speed; it may cause undesirable application behavior if balloon zeroing is done too aggressively; and convergence on an acceptable foreground synchronization time may not occur under many loads.

Balloon zeroing may be combined with other approaches. For example, the procedure 400 can be modified so that the target CE clears all of its memory prior to initiation of the background copy. In this case, only non-zero memory of the master CE would be copied during the first iteration of the background copy.

Hardware-Assisted Memory Controller Tracking Method

Referring to Fig. 6, with a small amount of hardware assistance from the memory controller chip, a simple design and implementation that increases the chances for convergence to an acceptable foreground wrap-up is possible. Rather than using the page table structure and internal knowledge of the operating system's use of PTEs and other mechanisms, a small but effective bitmap of block modifications can be maintained by the memory controller chip. The granularity of this bitmap is far less than what is obtainable with PTE tracking, but is adequate for the purpose of background synchronization. The procedure 600 differs from the procedure 100 shown in Fig. 1 only in that in step 605, which replaces step 115, the original memory copy list is stored using the memory controller chip, and in step 610, which replaces step 125, the updated memory copy list is created using the memory controller chip.

Advantages of this approach are that it is operating system independent, it is insensitive to operating system uses of page table structures, it is insensitive to operating system methods of process context switches, it is insensitive to future changes to operating system memory management and kernel, it provides compact and efficient scanning of bitmap results, the bitmap tracks all memory writes, including those originating from the processor and from the MIC, and convergence on an acceptable foreground synchronization time should be very high. This approach may be combined with balloon zeroing method to reduce background copy traffic. Potential drawbacks of this approach are that it does not work with existing systems, it is dependent on the motherboard and chip design cycle, its features are not supported on all future chipsets, and that the granularity of memory modifications is limited by bitmap size.

Hardware Approach to Background Synchronization

With the aid of specific features designed into the memory controller, a simple and effective background synchronization mechanism may be implemented. The memory controller provides a service whereby modifications made to areas of physical memory are flagged in a bitmap structure, preferably register-based. Software chooses the time to clear this bitmap and to enable the logging of memory writes to it. Software also requires the ability to select the resolution of the bitmap, expressed as the number of base 2 kilobytes represented by each bit. During the background transfer of CE memory, this bitmap is used to accumulate block-level modifications to main memory on the sourcing CE.

The size of the bitmap determines the tracking granularity, or resolution, of each bit. A 1024-bit bitmap covers the same 4 GB range with a resolution of 0.1%, or 4 MB per bit. The advantage is that automatic synchronization can be enabled at all times, allowing periodic attempts at CE synchronization, which has a high probability of success.

The minimum recommended size of a hardware-maintained bitmap is such that a 0.1% resolution is achievable. Table 2 lists the recommended bitmap sizes for various maximum configurations.

Maximum Memory Configuration of Memory Controller	Minimum Recommended Bitmap Size	Minimum Bitmap Resolution	Resolution as Percent of Maximum Memory
4 GB	1024 bits	4 MB/bit	0.1%
8 GB	2048 bits	4 MB/bit	0.1%
16 GB	4096 bits	4 MB/bit	0.1%
32 GB	8192 bits	4 MB/bit	0.1%
64 GB	16384 bits	4 MB/bit	0.1%

Table 2 - Recommended Bitmap Sizes

As indicated here, a bitmap of only 1024 bits is sufficient to support background synchronization on configurations with up to 4 GB of memory. This results in a worst-case resolution of 4MB, or 0.1% of the total memory, per bitmap bit. Allowing the resolution of the bitmap to be software settable allows smaller memory configurations to be tracked with comparable resolution.

Selectable resolutions finer than 1 MB/bit are necessary if the bitmap size implemented is larger than the appropriate minimum recommended size from Table 2. For instance, a bitmap of 2048 bits on a 4 GB (max) chipset requires a 0.5 MB/bit resolution to use all bitmap bits if only 1 GB is actually present.

This approach may be implemented in a system that provides a register-based bitmap of (at least) the recommended size (Table 2), or a memory-based bitmap using software-specified base address and range. This latter approach is preferred because it gives software control over resolution. Alternatively, a bitmap can be managed as a set of 32-bit registers, or as an internal or external RAM array which is private to the memory controller chip.

The system also allows the bitmap to be cleared by software, preferably using "longword writes" (i.e., overwriting the bitmap with words containing all zeroes).

The system also allows bitmap logging to be disabled (default) and enabled by software. Disabling and enabling does not alter the contents of the bitmap. When enabled, all processor or MIC originated writes to memory are tracked in the bitmap. Memory scrubbing operations performed by the memory controller itself are not tracked.

The system also allows the block size (resolution) of the bitmap to be software settable. Resolutions finer than 1 MB per bit are not essential, but are certainly desirable. If physical memory is used for the bitmap, software sets the base physical address of the map, along with the resolution (and, therefore, size).

Software guarantees that L1/L2 caches will be swept prior to evaluating the contents of the memory controller's bitmap. Software also guarantees that writes (clears) to the bitmap registers will never occur while bitmap tracking is enabled, ensuring that no dual-access hazards need to be arbitrated by the memory controller. Other software operations affecting a possible implementation, such as changing the base address of an external bitmap, occur only when the logging feature is disabled.

A number of embodiments of the invention have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the invention. Accordingly, other embodiments are within the scope of the following claims.

WHAT IS CLAIMED IS:

1. A method of synchronizing an inactive memory with an active memory in a fault-tolerant computer system that includes an active processor, the method comprising:

copying data from the active memory to the inactive memory using a background process that permits the active processor to perform normal operations while the copying is proceeding;

while the copying is proceeding, tracking regions of the active memory in which changes are made;

after copying is complete, determining whether data from the regions of the active memory in which changes were made can be copied to the inactive memory within a predetermined time period using a foreground process that prevents the active processor from performing normal operations;

if the data from the regions of the active memory in which changes were made can be copied to the inactive memory within the predetermined time period using the foreground process, copying the data from the regions of the active memory in which changes were made to the inactive memory using the foreground process; and

if the data from the regions of the active memory in which changes were made cannot be copied to the inactive memory within the predetermined time period using the foreground process, repeating the copying, tracking, and determining for the regions of the active memory in which changes were made.

2. The method of claim 1 further comprising, before copying data from the active memory to the inactive memory using the background process, evaluating whether the synchronizing is likely to be successful.

3. The method of claim 2 wherein evaluating whether the synchronizing is likely to be successful comprises comparing a rate at which data in the active memory are modified to a rate at which data can be transferred from the active memory to the inactive memory using the background process.

4. The method of claim 2 further comprising, when the result of the evaluating indicates that the synchronizing is not likely to be successful, increasing an amount of bandwidth allocated to the background process prior to repeating the copying, tracking, and determining for the regions of the active memory in which changes were made.

5. The method of claim 2 further comprising, when the result of the evaluating indicates that the synchronizing is not likely to be successful, restricting an amount of working memory for one or more running applications to a minimum amount that still permits the one or more running applications to run prior to repeating the copying, tracking, and determining for the regions of the active memory in which changes were made.

6. The method of claim 2 further comprising, when the result of the evaluating indicates that the synchronizing is not likely to be successful, performing a data compression operation on the data from the regions of active memory in which changes were made prior to repeating the copying, tracking, and determining for the regions of the active memory in which changes were made.

7. The method of claim 1 wherein the active memory is associated with the active processor.

8. The method of claim 1 wherein the active processor comprises a compute element and an I/O processor and the compute element implements the copying, tracking and determining.

9. The method of claim 1 further comprising creating a memory copy list identifying portions of the active memory for which data are to be copied to the inactive memory using the background process, wherein copying data using the background process comprises using the memory copy list.

10. The method of claim 9 wherein tracking regions of the active memory comprises creating a new memory copy list.

11. The method of claim 1 wherein the fault-tolerant system further comprises an inactive processor associated with the inactive memory, the active processor is associated with the active memory, and the method further comprises copying a context of the active processor to the inactive processor. .

12. The method of claim 1 wherein tracking regions of the active memory comprises using a page table structure comprising pages of memory and corresponding page table entries, with each page table entry including an indicator bit that is set when a memory location of the corresponding page of memory is modified.

13. The method of claim 1 wherein the active processor comprises a memory control section and tracking regions of the active memory comprises using a memory block structure allocated by the memory control section, including updating a flag corresponding to a block of memory whenever the block of memory is modified.

14. The method of claim 1 further comprising, when the data from the regions of the active memory in which changes were made cannot be copied to the inactive memory within the predetermined time period using the foreground process, increasing an amount of bandwidth allocated to the background process prior to repeating the copying, tracking, and determining for the regions of the active memory in which changes were made.

15. The method of claim 1 further comprising, when the data from the regions of the active memory in which changes were made cannot be copied to the inactive memory within the predetermined time period using the foreground process, restricting an amount of working memory for one or more running applications to a minimum amount that still permits the one or more running applications to run prior to repeating the copying, tracking, and determining for the regions of the active memory in which changes were made.

16. The method of claim 1 further comprising restricting an amount of working memory for one or more running applications to a minimum amount that still permits the one or more running applications to run prior to copying data from the active memory to the inactive memory using a background process.

17. The method of claim 1 further comprising, when the data from the regions of the active memory in which changes were made cannot be copied to the inactive memory within the predetermined time period using the foreground process, performing a data compression operation on the data from the regions of active memory in which changes were made prior to repeating the copying, tracking, and determining for the regions of the active memory in which changes were made.

18. The method of claim 1 further comprising performing a data compression operation on the data from the regions of active memory in which changes were made prior to copying data from the active memory to the inactive memory using a background process.

19. The method of claim 1, wherein the method is implemented by a synchronization process, the method further comprising allocating a number of pages of active memory to the synchronization process prior to copying data from the active memory to the inactive memory using a background process.

20. The method of claim 19 further comprising clearing pages of the inactive memory corresponding to the allocated pages of the active memory.

21. The method of claim 19 further comprising clearing all of the inactive memory.

22. The method of claim 1 further comprising:

clearing the inactive memory; and

determining which regions of the active memory contain nonzero data, and wherein the copying data from the active memory to the inactive memory using the background process further comprises copying only data from the regions of the active memory that contain nonzero data.

23. Software for use in synchronizing an inactive memory with an active memory in a fault-tolerant computer system that includes an active processor, the software residing on a computer-readable medium and comprising instructions causing the fault-tolerant computer system to:

copy data from the active memory to the inactive memory using a background process that permits the active processor to perform normal operations while the copying is proceeding;

while the copying is proceeding, track regions of the active memory in which changes are made;

after copying is complete, determine whether data from the regions of the active memory in which changes were made can be copied to the inactive memory within a predetermined time period using a foreground process that prevents the active processor from performing normal operations;

if the data from the regions of the active memory in which changes were made can be copied to the inactive memory within the predetermined time period using the foreground process, copy the data from the regions of the active memory in which changes were made to the inactive memory using the foreground process; and

if the data from the regions of the active memory in which changes were made cannot be copied to the inactive memory within the predetermined time period using the foreground process, repeat the copying, tracking, and determining for the regions of the active memory in which changes were made.

24. The software of claim 23, further comprising instructions causing the fault-tolerant computer system to evaluate whether the synchronizing is likely to be successful, prior to copying data from the active memory to the inactive memory using a background process.

25. The software of claim 23 wherein the fault-tolerant computer system further comprises an inactive processor associated with the inactive memory, the active processor is

associated with the active memory, and the software further comprises instructions for causing the system to copy a context of the active processor to the inactive processor.

26. The software of claim 23 wherein instructions causing the system to track regions of the active memory comprise instructions causing the system to use a page table structure including pages of memory and corresponding page table entries, with each page table entry including an indicator bit that is set when a memory location of the corresponding page of memory is modified.

27. The software of claim 23 wherein the active processor comprises a memory control section and instructions causing the system to track regions of the active memory comprise instructions causing the system to use a memory block structure allocated by the memory control section, including instructions causing the system to update a flag corresponding to a block of memory whenever the block of memory is modified.

28. The software of claim 23 further comprising, when the data from the regions of the active memory in which changes were made cannot be copied to the inactive memory within the predetermined time period using the foreground process, instructions causing the system to increase an amount of bandwidth allocated to the background process prior to repeating the copying, tracking, and determining for the regions of the active memory in which changes were made.

29. The software of claim 23 further comprising instructions causing the system to restrict an amount of working memory for one or more running applications to a minimum amount that still permits the one or more running applications to run prior to copying data from the active memory to the inactive memory using a background process.

30. The software of claim 23 further comprising instructions causing the system to perform a data compression operation on the data from the regions of active memory in which changes were made prior to copying data from the active memory to the inactive memory using a background process.

31. The software of claim 23 further comprising instructions for causing the system to allocate a number of pages of active memory to a synchronization process prior to copying data from the active memory to the inactive memory using a background process.

32. The software of claim 23 further comprising instructions for causing the system to:

clear the inactive memory; and

determine which regions of the active memory contain nonzero data, and

wherein the instructions for causing the system to copy data from the active memory to the inactive memory using the background process further comprise instructions for causing the system to copy only data from the regions of the active memory that contain nonzero data.

33. A fault-tolerant computer system comprising an active processor with associated active memory and an inactive processor with associated inactive memory, the system configured to synchronize the inactive memory with the active memory by: copying data from the active memory to the inactive memory using a background process that permits the active processor to perform normal operations while the copying is proceeding;

while the copying is proceeding, tracking regions of the active memory in which changes are made;

after copying is complete, determining whether data from the regions of the active memory in which changes were made can be copied to the inactive memory within a predetermined time period using a foreground process that prevents the active processor from performing normal operations;

if the data from the regions of the active memory in which changes were made can be copied to the inactive memory within the predetermined time period using the foreground process, copying the data from the regions of the active memory in which changes were made to the inactive memory using the foreground process; and

if the data from the regions of the active memory in which changes were made cannot be copied to the inactive memory within the predetermined time period using the foreground process, repeating the copying, tracking, and determining for the regions of the active memory in which changes were made.

34. The fault-tolerant computer system of claim 33, the system being further configured to evaluate whether the synchronizing is likely to be successful before copying data from the active memory to the inactive memory using the background process.

35. The fault-tolerant computer system of claim 33, the system being further configured to copy a context of the active processor to the inactive processor.

36. The fault-tolerant computer system of claim 33 wherein tracking regions of the active memory comprises using a page table structure including pages of memory and corresponding page table entries, with each page table entry including an indicator bit that is set when a memory location of the corresponding page of memory is modified.

37. The fault-tolerant computer system of claim 33 wherein the active processor comprises a memory control section and tracking regions of the active memory comprises using a memory block structure allocated by the memory control section, including updating a flag corresponding to a block of memory whenever the block of memory is modified.

38. The fault-tolerant computer system of claim 33, the system being further configured to increase an amount of bandwidth allocated to the background process prior to repeating the copying, tracking, and determining for the regions of the active memory in which changes were made when the data from the regions of the active memory in which changes were made cannot be copied to the inactive memory within the predetermined time period using the foreground process.

39. The fault-tolerant computer system of claim 33, the system being further configured to restrict an amount of working memory for one or more running applications to a minimum amount that still permits the one or more running applications to run prior to copying data from the active memory to the inactive memory using a background process.

40. The fault-tolerant computer system of claim 33, the system being further configured to perform a data compression operation on the data from the regions of active

memory in which changes were made prior to copying data from the active memory to the inactive memory using a background process.

41. The fault-tolerant computer system of claim 33, the system being further configured to allocate a number of pages of active memory to a synchronization process prior to copying data from the active memory to the inactive memory using a background process.

42. The fault-tolerant computer system of claim 33, the system being further configured to:

clear the inactive memory; and

determine which regions of the active memory contain nonzero data, and

wherein copying data from the active memory to the inactive memory using the background process further comprises copying only data from the regions of the active memory that contain nonzero data.

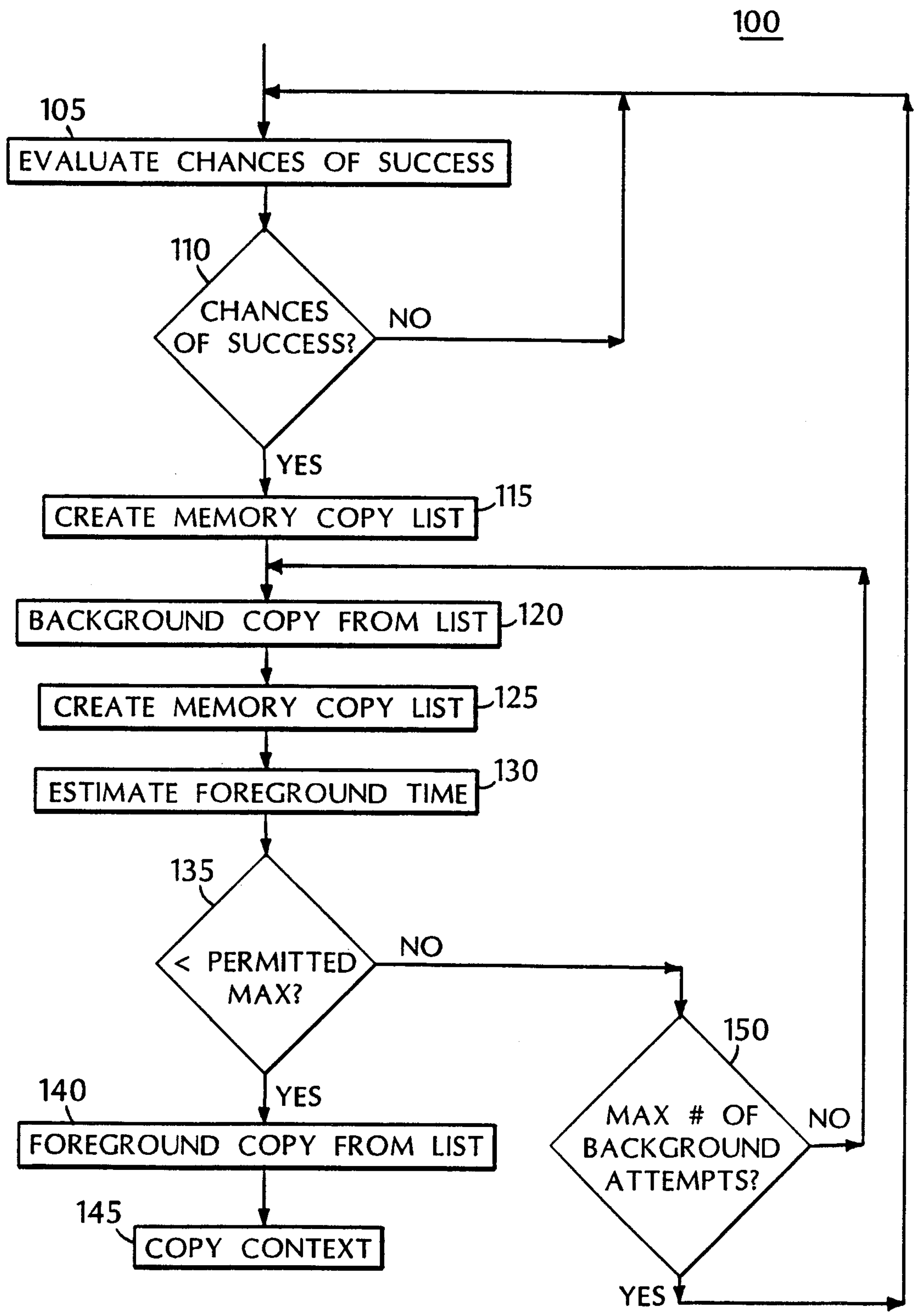

100
105
EVALUATE CHANCES OF SUCCESS
110
CHANCES OF SUCCESS?
NO
YES
CREATE MEMORY COPY LIST
115
BACKGROUND COPY FROM LIST
120
CREATE MEMORY COPY LIST
125
ESTIMATE FOREGROUND TIME
130
135
< PERMITTED MAX?
NO
YES
140
FOREGROUND COPY FROM LIST
145
COPY CONTEXT
150
MAX # OF BACKGROUND ATTEMPTS?
NO
YES

FIG. 1

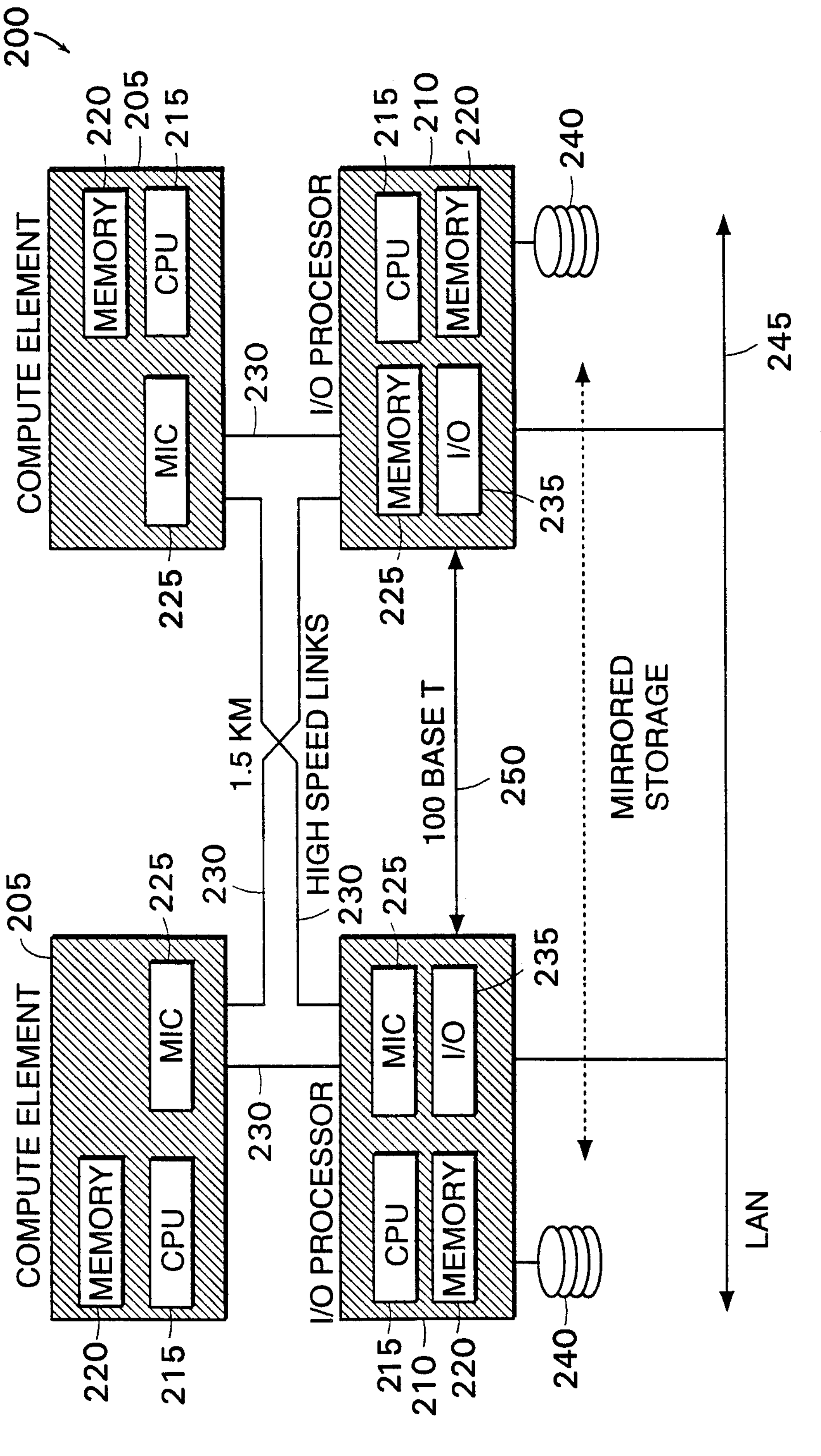

200
COMPUTE ELEMENT
MEMORY
CPU
MIC
220
205
215
225
230
I/O PROCESSOR
CPU
MEMORY
MEMORY
I/O
210
235
240
245
1.5 KM
HIGH SPEED LINKS
100 BASE T
250
MIRRORED STORAGE
LAN

FIG. 2

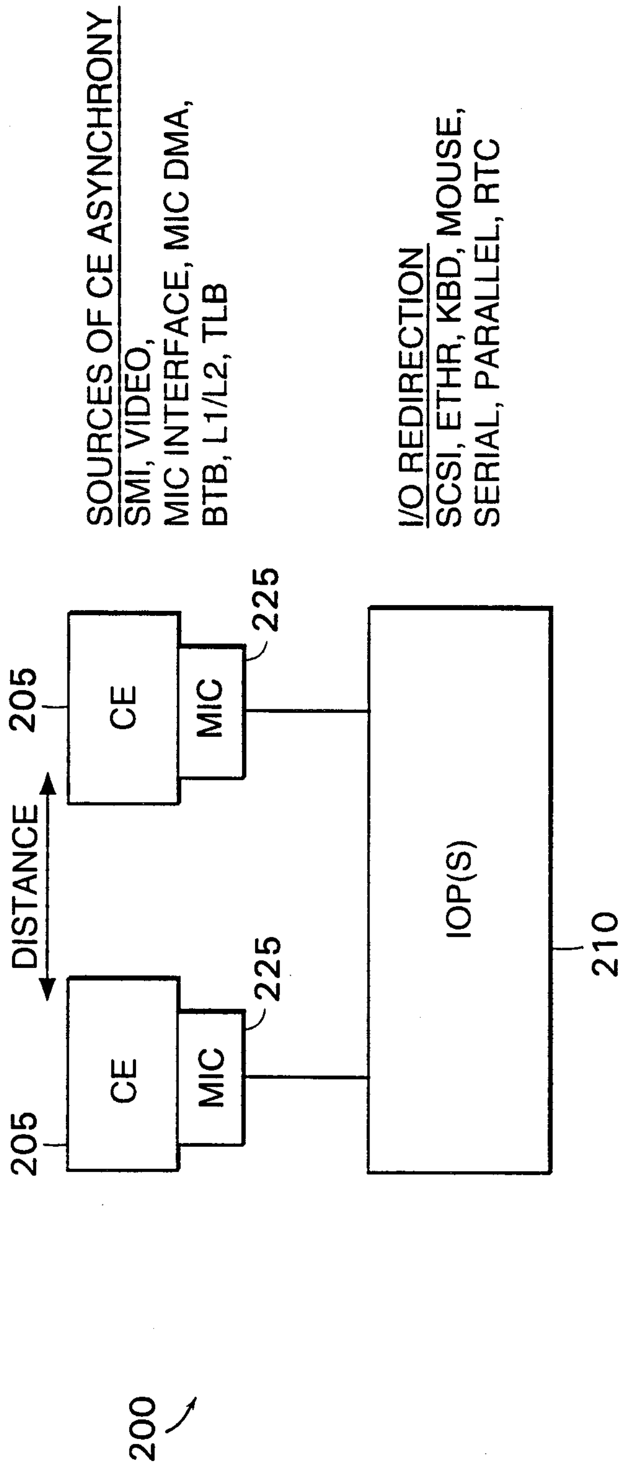

SOURCES OF CE ASYNCHRONY
SMI, VIDEO,
MIC INTERFACE, MIC DMA,
BTB, L1/L2, TLB
I/O REDIRECTION
SCSI, ETHR, KBD, MOUSE,
SERIAL, PARALLEL, RTC
205
CE
MIC
225
DISTANCE
205
CE
MIC
225
IOP(S)
210
200

FIG. 3

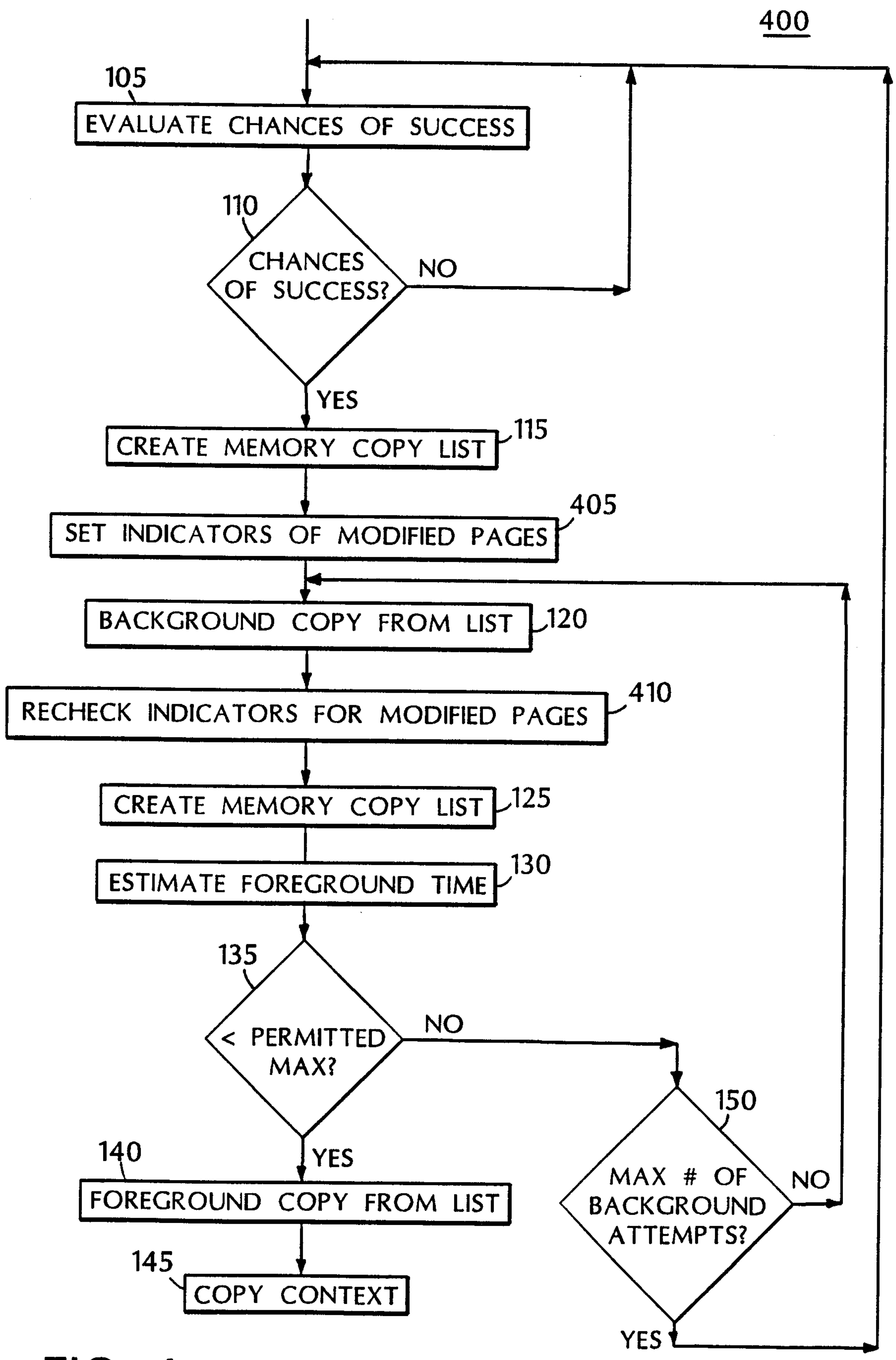

400
105
EVALUATE CHANCES OF SUCCESS
110
CHANCES OF SUCCESS?
NO
YES
CREATE MEMORY COPY LIST
115
SET INDICATORS OF MODIFIED PAGES
405
BACKGROUND COPY FROM LIST
120
RECHECK INDICATORS FOR MODIFIED PAGES
410
CREATE MEMORY COPY LIST
125
ESTIMATE FOREGROUND TIME
130
135
< PERMITTED MAX?
NO
YES
140
FOREGROUND COPY FROM LIST
145
COPY CONTEXT
150
MAX # OF BACKGROUND ATTEMPTS?
NO
YES

FIG. 4

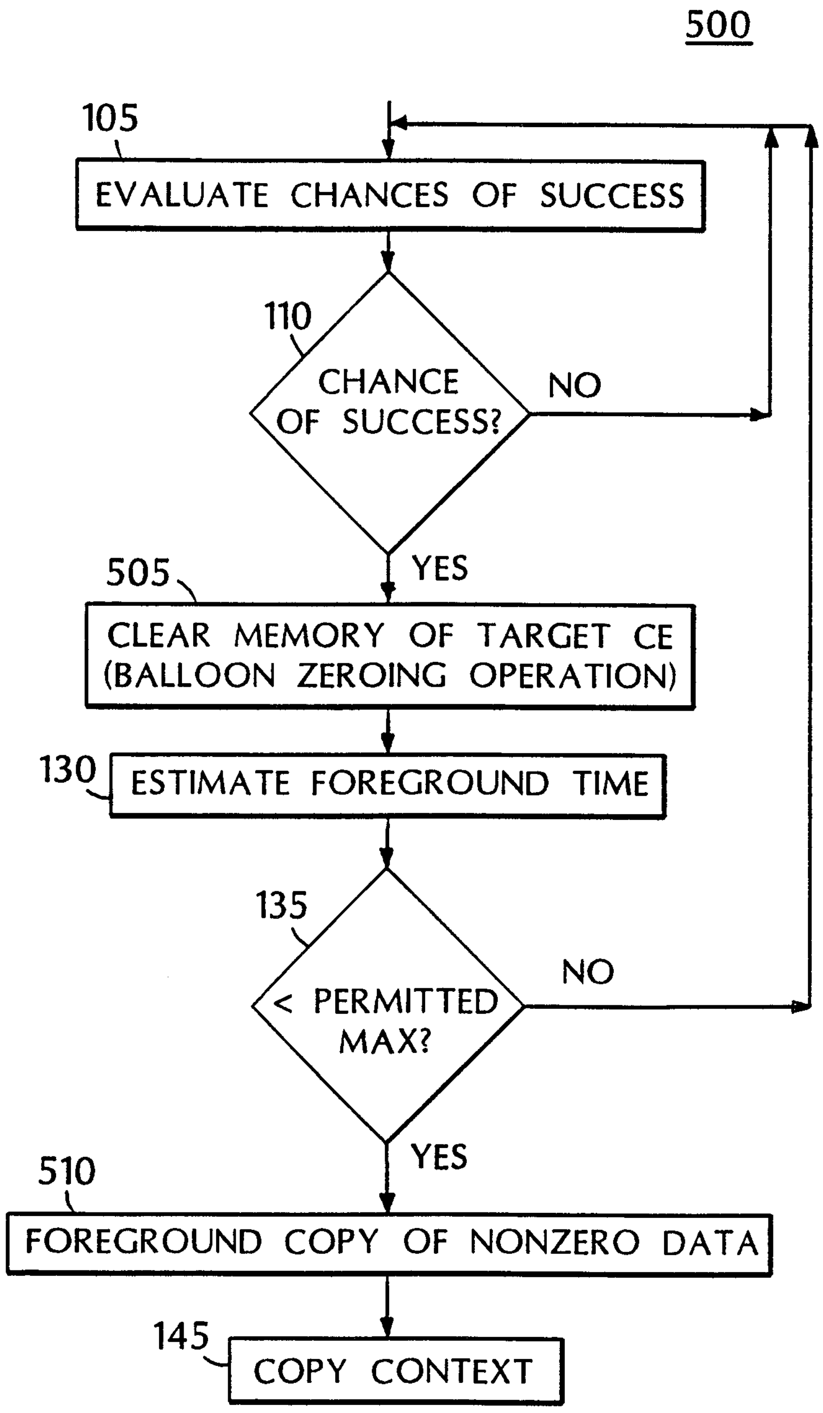

500
105
EVALUATE CHANCES OF SUCCESS
110
CHANCE OF SUCCESS?
NO
YES
505
CLEAR MEMORY OF TARGET CE (BALLOON ZEROING OPERATION)
130
ESTIMATE FOREGROUND TIME
135
< PERMITTED MAX?
NO
YES
510
FOREGROUND COPY OF NONZERO DATA
145
COPY CONTEXT

FIG. 5

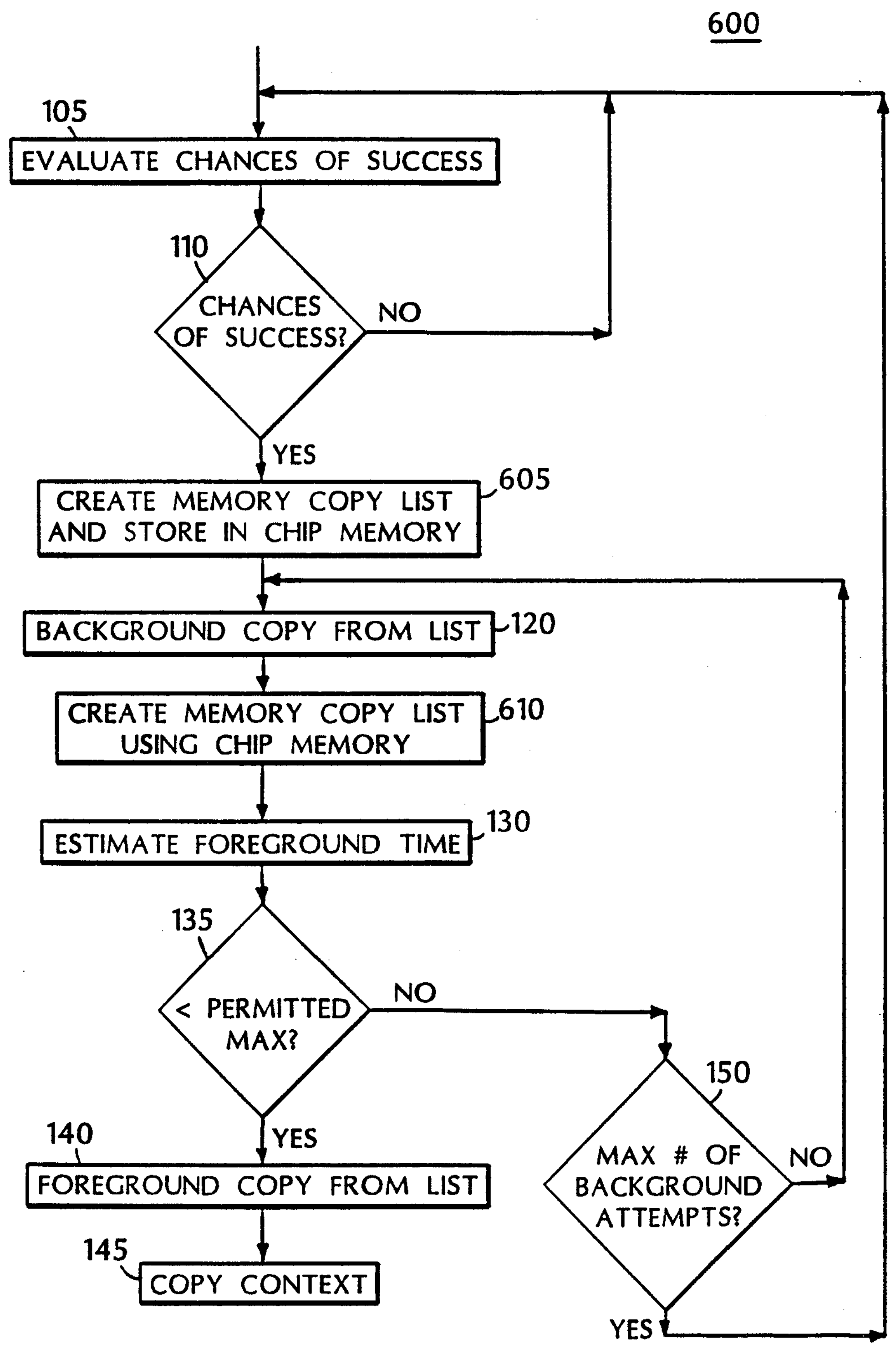

600
105
EVALUATE CHANCES OF SUCCESS
110
CHANCES OF SUCCESS?
NO
YES
CREATE MEMORY COPY LIST AND STORE IN CHIP MEMORY
605
BACKGROUND COPY FROM LIST
120
CREATE MEMORY COPY LIST USING CHIP MEMORY
610
ESTIMATE FOREGROUND TIME
130
135
< PERMITTED MAX?
NO
YES
150
MAX # OF BACKGROUND ATTEMPTS?
NO
YES
140
FOREGROUND COPY FROM LIST
145
COPY CONTEXT

FIG. 6

INTERNATIONAL SEARCH REPORT

Interr nal Application No
PCT/US 00/08940

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F11/14 G06F11/20

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 579 220 A (BARTHEL HERBERT ET AL) 26 November 1996 (1996-11-26) column 1, line 61 -column 4, line 28; figure 1	1,2,7-9, 11, 23-25, 33-35
A	US 5 608 865 A (HOLBERGER KENNETH D ET AL) 4 March 1997 (1997-03-04) column 22, line 9 - line 41	1,2,7-9, 11, 23-25, 33-35

☐ Further documents are listed in the continuation of box C.

[✓ X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier document but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
"&" document member of the same patent family

Date of the actual completion of the international search 12 July 2000	Date of mailing of the international search report 18/07/2000
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Authorized officer Fernandez Balseiro,J

2

INTERNATIONAL SEARCH REPORT

Information on patent family members

Intern al Application No
PCT/US 00/08940

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5579220 A	26-11-1996	AT 180902 T	15-06-1999
		DE 59408335 D	08-07-1999
		EP 0636956 A	01-02-1995
		ES 2134883 T	16-10-1999
US 5608865 A	04-03-1997	NONE	