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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 November 2004 (04.11.2004)

PCT

(10) International Publication Number
WO 2004/095271 A2

(51) International Patent Classification[7]: G06F 9/45

(21) International Application Number: PCT/US2004/007649

(22) International Filing Date: 11 March 2004 (11.03.2004)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
10/400,224 26 March 2003 (26.03.2003) US

(71) Applicant *(for all designated States except US)*: **SUN MICROSYSTEMS, INC.** [US/US]; 4120 Network Circle, MS SCA 12-203, Santa Clara, CA 95054 (US).

(71) Applicants and
(72) Inventors: **MOLCHANOV, Nikolay** [RU/US]; 41238 Robert Avenue, #5, Fremont, CA 94538 (US). **PRAKASH, Raj** [US/US]; 20585 Carniel Avenue, Saratoga, CA 95070 (US).

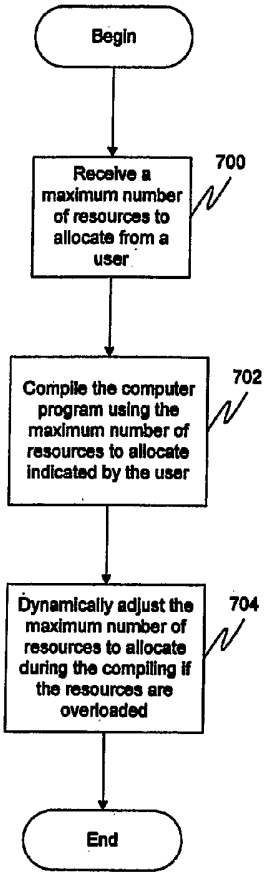
(74) Agents: **RITCHIE, David, B.** et al.; Thelen Reid & Priest, LLP, P.O. Box 640640, San Jose, CA 95164-0640 (US).

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

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, 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, IT, LU, MC, NL, PL, PT, RO, SE, SI, SK,

[Continued on next page]

(54) Title: DYNAMIC DISTRIBUTED MAKE



(57) Abstract: A distributed make command is used when compiling a computer program in order to allow non-dependent processes in the compiling be performed in parallel, such as with different resources, in order to speed compile time. The distributed make command is typically executed by a user who also specifies a maximum number of resources to allocate to the compiling. The present invention dynamically adjusts this maximum number of resources to allocate if the resources become overloaded during the compiling, which has the effect of optimizing the efficiency of the compiling, either by reducing the number of resources utilized or by reducing the amount of time the compiling takes (or both).

WO 2004/095271 A2

TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report*

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.

SPECIFICATION

TITLE OF INVENTION

DYNAMIC DISTRIBUTED MAKE

FIELD OF THE INVENTION

[0001] The present invention relates to the field of computer software compiling. More particularly, the present invention relates to a dynamic distributed make for computer software compiling.

BACKGROUND OF THE INVENTION

[0002] When a computer program is built, the code is typically written in a high level language, such as C++. The code is typically placed in one or more source files. Upon compiling, these source files may be made into object files, and combined with other object files to create the final software.

[0003] Make is a software tool that automates the procedures required to compile a program. The interdependencies between different source files may be stored in a makefile, which enables Make to recompile only those files that have been changed. A typical makefile may read as follows:

```
x1.o:
        cc -c x1.c
x2.o:
        cc -c  x2.c
a.out: x1.o  x2.o
        ld -o a.out x1.o x2.o
```

[0004] This indicates that object file x1.o is made by compiling x1.c, x2.o is made by compiling x2.c, and a.out is made by linking x1.o and x2.o.

[0005] Make operates sequentially, as due to the dependencies certain actions must be done before other actions. However, there are some situations where actions could be done simultaneously (if they are not dependent on one another). Make does not provide for any facility to take advantage of this fact. Distributed Make (Dmake) was therefore created to allow non-dependent actions to be executed in parallel, such as on different systems, to speed compile time.

[0006] The Dmake command includes a parameter indicating a maximum number of resources to utilize. This is often used because it is can be beneficial to leave resources for other people.

[0007] The problem with this maximum parameter is that it can often be set too high for optimal compiling. This occurs if, for example, the programmer overestimates the amount of resources available or unexpected jobs begin utilizing resources.

[0008] This problem may be illustrated through the use of several examples. In each of these examples, Dmake starts 64 parallel jobs, with each job taking 256 MB of memory and 60 seconds of CPU time. FIG. 1 is a graph illustrating an example of the build operation in a system having 64 CPUs and more than 16GB of memory. This is a case where there are plenty of resources, and thus the program compiles quickly, in about 60 seconds.

[0009] FIG. 2 is a graph illustrating an example of the build operation in a system having 4 CPUs and more than 16 GB of memory. Here, there are a moderate amount of resources, and thus the program still compiles in a reasonable time, about 20 minutes, although not as fast as the example in FIG. 1.

[0010] FIG. 3 is a graph illustrating an example of the build operation in a system having 4 CPIs and less than 16GB of memory. Here, the build will take a very long time because when the memory limit is exceeded, swapping must occur (indicated by area 300). The overall parallel system runs very inefficiently whenever swapping must occur.

[0011] Thus, the maximum parameter specified by the programmer may result in an oversubscription of the system, resulting in a less than ideal throughput. What is needed is a solution that achieves the optimum throughput on the system.

BRIEF DESCRIPTION

[0012] A distributed make command is used when compiling a computer program in order to allow non-dependent processes in the compiling be performed in parallel, such as with different resources, in order to speed compile time. The distributed make command is typically executed by a user who also specifies a maximum number of resources to allocate to the compiling. The present invention dynamically adjusts this maximum number of resources to allocate if the resources become overloaded during the compiling, which has the effect of optimizing the efficiency of the compiling, either by reducing the number of resources utilized or by reducing the amount of time the compiling takes (or both).

BRIEF DESCRIPTION OF THE DRAWINGS

[0013] The accompanying drawings, which are incorporated into and constitute a part of this specification, illustrate one or more embodiments of the present invention and, together with the detailed description, serve to explain the principles and implementations of the invention.

[0014] In the drawings:

FIG. 1 is a graph illustrating an example of the build operation in a system having 64 CPUs and more than 16GB of memory.

FIG. 2 is a graph illustrating an example of the build operation in a system having 4 CPUs and more than 16 GB of memory.

FIG. 3 is a graph illustrating an example of the build operation in a system having 4 CPIs and less than 16GB of memory.

FIG. 4 is a graph illustrating an example of the build operation in a system having 64 CPUs and more than 16GB of memory in accordance with an embodiment of the present invention.

FIG. 5 is a graph illustrating an example of the build operation in a system having 4 CPUs and more than 16 GB of memory in accordance with an embodiment of the present invention.

FIG. 6 is a graph illustrating an example of the build operation in a system having 4 CPIs and less than 16GB of memory in accordance with an embodiment of the present invention.

FIG. 7 is a flow diagram illustrating a method for optimizing resource utilized in a compiling a computer program in accordance with an embodiment of the present invention.

FIG. 8 is a flow diagram illustrating a method for optimizing resource utilized in a compiling a computer program in accordance with another embodiment of the present invention.

FIG. 9 is a flow diagram illustrating a method for executing a distributed make command having a maximum number of resources to allocate as a parameter in accordance with an of the present invention.

FIG. 10 is a block diagram illustrating an apparatus for optimizing resource utilized in a compiling a computer program in accordance with an embodiment of the present invention.

FIG. 11 is a block diagram illustrating an apparatus for optimizing resource utilized in a compiling a computer program in accordance with another embodiment of the present invention.

FIG. 12 is a block diagram illustrating an apparatus for executing a distributed make command having a maximum number of resources to allocate as a parameter in accordance with an of the present invention.

DETAILED DESCRIPTION

[0015] Embodiments of the present invention are described herein in the context of a system of computers, servers, and software. Those of ordinary skill in the art will realize that the following detailed description of the present invention is illustrative only and is not intended to be in any way limiting. Other embodiments of the present invention will readily suggest themselves to such skilled persons having the benefit of this disclosure. Reference will now be made in detail to implementations of the present invention as illustrated in the accompanying drawings. The same reference indicators will be used throughout the drawings and the following detailed description to refer to the same or like parts.

[0016] In the interest of clarity, not all of the routine features of the implementations described herein are shown and described. It will, of course, be appreciated that in the development of any such actual implementation, numerous implementation-specific decisions must be made in order to achieve the developer's specific goals, such as compliance with application- and business-related constraints, and that these specific goals will vary from one implementation to another and from one developer to another. Moreover, it will be appreciated that such a development effort might be complex and time-consuming, but would nevertheless be a routine undertaking of engineering for those of ordinary skill in the art having the benefit of this disclosure.

[0017] In accordance with the present invention, the components, process steps, and/or data structures may be implemented using various types of operating systems, computing platforms, computer programs, and/or general purpose machines. In addition, those of ordinary skill in the art will recognize that devices of a less general purpose nature, such as hardwired devices, field programmable gate arrays (FPGAs), application specific integrated circuits (ASICs), or the like, may also be used without departing from the scope and spirit of the inventive concepts disclosed herein.

[0018] The present invention achieves optimal throughout on a parallel computing system by providing a mechanism to dynamically adjust the number of parallel processes that are spawned based on available resources. The maximum parameter specified by the programmer may be modified as the compile is occurring. If the system is not overloaded, Dmake may continue to spawn new parallel processes while keeping track of

the load on the system. When the system becomes sufficiently busy, dmake may set the limit of the allowed parallel jobs to a lower value.

[0019] Thus, in one embodiment of the present invention, the adjusted limit of processes (N) may be set equal to the maximum limit set by the programmer (L) minus a function involving the total number of active processes in the operating system (Q) over the number of CPUs (C). Thus, $N = L$ - func(Q/C), or to cover the case where it may need to be returned back to the original limit when resources are no longer overloaded, N = MAX (1, L - func (Q/C)).

[0020] In another embodiment of the present invention, the adjusted limit of processes may be set via a more specific function, such as N = MAX (1, L - Q/C).

[0021] These embodiments should not be read as limiting, as the functions may be set as appropriate for the particular system and type of resources involved. For example, there is no requirement that the number of avaialble CPUs be utilized as the sole measure of available resources - other factors may be utilized as well. Additionally, it may be beneficial in certain instances to wait to apply the function until the system is significantly overloaded. Thus, in the embodiments described above, rather than adjusting the limit as soon as Q exceeds C, it may be more beneficial to wait until Q exceeds C by 200 or 300 percent.

[0022] While this solution can reduce the total amount of compiling time required for a project, it is also beneficial in that it can often reduce the number of resources utilized while not impacting the total compiling time. The following examples may be utilized to illustrate this.

[0023] FIG. 4 is a graph illustrating an example of the build operation in a system having 64 CPUs and more than 16GB of memory in accordance with an embodiment of the present invention. This is a case like FIG. 1 where there are plenty of resources, and thus the program compiles quickly, in about 60 seconds. Thus, while there is no advantage in this example to using an embodiment of the present invention, there is no disadvantage either.

[0024] FIG. 5 is a graph illustrating an example of the build operation in a system having 4 CPUs and more than 16 GB of memory in accordance with an embodiment of the present invention. Here, there are a moderate amount of resources, similar to FIG. 2, and the results from FIG. 2 are illustrated by line 500. Line 502, however, indicates the results using an embodiment of the present invention. As can be seen, the compiling still takes 20 minutes, but the number of resources utilized is much improved over the example in FIG. 2.

[0025] FIG. 6 is a graph illustrating an example of the build operation in a system having 4 CPIs and less than 16GB of memory in accordance with an embodiment of the present invention. Area 600 indicates a region where swapping must occur because of a memory overloading problem. Here, the results from FIG. 3 are illustrated by line 602. Line 604, however, indicates the results using an embodiment of the present invention. As can be seen, the embodiment of the present invention avoids overloading the memory, resulting in both fewer resources being utilized and the compiling occurring in a reasonable time (20 minutes).

[0026] FIG. 7 is a flow diagram illustrating a method for optimizing resource utilized in a compiling a computer program in accordance with an embodiment of the present invention. At 700, a maximum number of resources to allocate may be received from a user. At 702, the computer program may be compiled using the maximum number of resources to allocate indicated by the user. At 704, the maximum number of resources to allocate may be dynamically adjusted during the compiling if the resources are overloaded. Dynamically adjusting may comprise, for example, lowering the maximum number of resources by a function of the number of active processes divided by the number of resources. It may also comprise, for example, setting the maximum number of resources equal to the maximum of either one or a value equal to the maximum number of resources to allocate indicated by the user minus a function of the number of active processes divided by the number of resources. The resources may be considered overloaded if, for example, the number of active processes exceeds the number of active resources, exceeds it by at least 200%, or exceeds it by at least 300%.

[0027] FIG. 8 is a flow diagram illustrating a method for optimizing resource utilized in a compiling a computer program in accordance with another embodiment of the

present invention. At 800, a maximum number of resources to allocate may be received from a user. At 802, at least one process may be executed to compile the computer program. For each process, the execution may be delayed at 804 if the maximum number of resources to allocate has been exceeded by a number of active processes. At 806, the maximum number of resources to allocate may be dynamically adjusted if the resources are overloaded. Dynamically adjusting may comprise, for example, lowering the maximum number of resources by a function of the number of active processes divided by the number of resources. It may also comprise, for example, setting the maximum number of resources equal to the maximum of either one or a value equal to the maximum number of resources to allocate indicated by the user minus a function of the number of active processes divided by the number of resources. The resources may be considered overloaded if, for example, the number of active processes exceeds the number of active resources, exceeds it by at least 200%, or exceeds it by at least 300%. At 808, the process may be executed.

[0028] FIG. 9 is a flow diagram illustrating a method for executing a distributed make command having a maximum number of resources to allocate as a parameter in accordance with an of the present invention. At 900, at least one process specified in a makefile may be executed to compile the computer program. For each process, the execution may be delayed at 902 if the maximum number of resources to allocate has been exceeded by a number of active processes. At 904, the maximum number of resources to allocate may be dynamically adjusted if the resources are overloaded. Dynamically adjusting may comprise, for example, lowering the maximum number of resources by a function of the number of active processes divided by the number of resources. It may also comprise, for example, setting the maximum number of resources equal to the maximum of either one or a value equal to the maximum number of resources to allocate indicated by the user minus a function of the number of active processes divided by the number of resources. The resources may be considered overloaded if, for example, the number of active processes exceeds the number of active resources, exceeds it by at least 200%, or exceeds it by at least 300%. At 906, the process may be executed.

[0029] FIG. 11 is a block diagram illustrating an apparatus for optimizing resource utilized in a compiling a computer program in accordance with another embodiment of

the present invention. A maximum number of resources receiver 1100 may receive a maximum number of resources to allocate from a user. A process executor 1102 coupled to the maximum number of resources receiver may execute at least one process to compile the computer program. For each process, the execution may be delayed if the maximum number of resources to allocate has been exceeded by a number of active processes. The maximum number of resources to allocate may be dynamically adjusted if the resources are overloaded. Dynamically adjusting may comprise, for example, lowering the maximum number of resources by a function of the number of active processes divided by the number of resources. It may also comprise, for example, setting the maximum number of resources equal to the maximum of either one or a value equal to the maximum number of resources to allocate indicated by the user minus a function of the number of active processes divided by the number of resources. The resources may be considered overloaded if, for example, the number of active processes exceeds the number of active resources, exceeds it by at least 200%, or exceeds it by at least 300%. The process may then be executed.

[0030] FIG. 12 is a block diagram illustrating an apparatus for executing a distributed make command having a maximum number of resources to allocate as a parameter in accordance with an of the present invention. A makefile process extractor 1200 may execute at least one process specified in a makefile to compile the computer program. For each process, the execution may be delayed if the maximum number of resources to allocate has been exceeded by a number of active processes. The maximum number of resources to allocate may be dynamically adjusted if the resources are overloaded. Dynamically adjusting may comprise, for example, lowering the maximum number of resources by a function of the number of active processes divided by the number of resources. It may also comprise, for example, setting the maximum number of resources equal to the maximum of either one or a value equal to the maximum number of resources to allocate indicated by the user minus a function of the number of active processes divided by the number of resources. The resources may be considered overloaded if, for example, the number of active processes exceeds the number of active resources, exceeds it by at least 200%, or exceeds it by at least 300%. The process may then be executed.

[0031] While embodiments and applications of this invention have been shown and described, it would be apparent to those skilled in the art having the benefit of this disclosure that many more modifications than mentioned above are possible without departing from the inventive concepts herein. The invention, therefore, is not to be restricted except in the spirit of the appended claims.

CLAIMS

What is claimed is:

1. A method for optimizing resources utilized in compiling a computer program, the method comprising:

receiving a maximum number of resources to allocate from a user;

compiling the computer program using said maximum number of resources to allocate indicated by said user; and

dynamically adjusting said maximum number of resources to allocate during said compiling if said resources are overloaded.

2. The method of claim 1, wherein said dynamically adjusting comprises lowering said maximum number of resources by a function of the number of active processes divided by the number of resources.

3. The method of claim 1, wherein said resources are overloaded if the number of active processes exceeds the number of resources.

4. The method of claim 1, wherein said resources are overloaded if the number of active processes exceeds the number of resources by 200% or more.

5. The method of claim 1, wherein said resources are overloaded if the number of active processes exceeds the number of resources by 300% or more.

6. The method of claim 1, wherein said dynamically adjusting includes setting said maximum number of resources equal to the maximum of either one or a value equal to the maximum number of resources to allocate indicated by said user minus a function of the number of active processes divided by the number of resources.

7. A method for optimizing resource utilized in compiling a computer program, the method comprising:

receiving a maximum number of resources to allocate from a user;

executing at least one process to compile said computer program, said executing further including for each process:

delaying executing said process if said maximum number of resources to allocate has been exceeded by a number of active processes;
dynamically adjusting said maximum number of resources to allocate if said resources are overloaded; and executing said process.

8. The method of claim 7, wherein said dynamically adjusting comprises lowering said maximum number of resources by a function of the number of active processes divided by the number of resources.

9. The method of claim 7, wherein said resources are overloaded if the number of active processes exceeds the number of resources.

10. The method of claim 7, wherein said resources are overloaded if the number of active processes exceeds the number of resources by 200% or more.

11. The method of claim 7, wherein said resources are overloaded if the number of active processes exceeds the number of resources by 300% or more.

12. The method of claim 7, wherein said dynamically adjusting includes setting said maximum number of resources equal to the maximum of either one or a value equal to the maximum number of resources to allocate indicated by said user minus a function of the number of active processes divided by the number of resources.

13. A method for executing a distributed make command having a maximum number of resources to allocate as a parameter, the method comprising:
executing at least one process specified in a make file to compile said computer program, said executing further including for each process:
delaying executing said process if said maximum number of resources to allocate has been exceeded by a number of active processes;
dynamically adjusting said maximum number of resources to allocate if said resources are overloaded; and executing said process.

14. The method of claim 13, wherein said dynamically adjusting comprises lowering said maximum number of resources by a function of the number of active processes divided by the number of resources.

15. The method of claim 13, wherein said resources are overloaded if the number of active processes exceeds the number of resources.

16. The method of claim 13, wherein said resources are overloaded if the number of active processes exceeds the number of resources by 200% or more.

17. The method of claim 13, wherein said resources are overloaded if the number of active processes exceeds the number of resources by 300% or more.

18. The method of claim 13, wherein said dynamically adjusting includes setting said maximum number of resources equal to the maximum of either one or a value equal to the maximum number of resources to allocate indicated by said user minus a function of the number of active processes divided by the number of resources.

19. An apparatus for optimizing resources utilized in compiling a computer program, the apparatus comprising:

a maximum number of resources receiver;

a computer program compiler coupled to said maximum number of resources receiver;

a maximum number of resources dynamic adjuster coupled to said computer program compiler.

20. An apparatus for optimizing resource utilized in compiling a computer program, the apparatus comprising:

a maximum number of resources receiver; and

a process executor coupled to said maximum number of resources receiver, said process executor including a process execution delayer and a maximum number of resources dynamic adjuster.

21. An apparatus for executing a distributed make command having a maximum number of resources to allocate as a parameter, the apparatus comprising:

a makefile process executor, said makefile process executor including: a process execution delayer and a maximum number of resources dynamic adjuster.

22. An apparatus for optimizing resources utilized in compiling a computer program, the apparatus comprising:

means for receiving a maximum number of resources to allocate from a user;

means for compiling the computer program using said maximum number of resources to allocate indicated by said user; and

means for dynamically adjusting said maximum number of resources to allocate during said compiling if said resources are overloaded.

23. The apparatus of claim 22, wherein said means for dynamically adjusting comprises means for lowering said maximum number of resources by a function of the number of active processes divided by the number of resources.

24. The apparatus of claim 22, wherein said resources are overloaded if the number of active processes exceeds the number of resources.

25. The apparatus of claim 22, wherein said resources are overloaded if the number of active processes exceeds the number of resources by 200% or more.

26. The apparatus of claim 22, wherein said resources are overloaded if the number of active processes exceeds the number of resources by 300% or more.

27. The apparatus of claim 22, wherein said means for dynamically adjusting includes means for setting said maximum number of resources equal to the maximum of either one or a value equal to the maximum number of resources to allocate indicated by said user minus a function of the number of active processes divided by the number of resources.

28. An apparatus for optimizing resource utilized in compiling a computer program, the apparatus comprising:

means for receiving a maximum number of resources to allocate from a user;
means for executing at least one process to compile said computer program, said means for executing further including for each process:
means for delaying executing said process if said maximum number of resources to allocate has been exceeded by a number of active processes;
means for dynamically adjusting said maximum number of resources to allocate if said resources are overloaded; and means for executing said process.

29. The apparatus of claim 28, wherein said means for dynamically adjusting comprises means for lowering said maximum number of resources by a function of the number of active processes divided by the number of resources.

30. The apparatus of claim 28, wherein said resources are overloaded if the number of active processes exceeds the number of resources.

31. The apparatus of claim 28, wherein said resources are overloaded if the number of active processes exceeds the number of resources by 200% or more.

32. The apparatus of claim 28, wherein said resources are overloaded if the number of active processes exceeds the number of resources by 300% or more.

33. The apparatus of claim 28, wherein said means for dynamically adjusting includes means for setting said maximum number of resources equal to the maximum of either one or a value equal to the maximum number of resources to allocate indicated by said user minus a function of the number of active processes divided by the number of resources.

34. A program storage device readable by a machine, tangibly embodying a program of instructions executable by the machine to perform a method for optimizing resources utilized in compiling a computer program, the method comprising:

receiving a maximum number of resources to allocate from a user;

compiling the computer program using said maximum number of resources to allocate indicated by said user; and

dynamically adjusting said maximum number of resources to allocate during said compiling if said resources are overloaded.

35. A program storage device readable by a machine, tangibly embodying a program of instructions executable by the machine to perform a method for optimizing resource utilized in compiling a computer program, the method comprising:
receiving a maximum number of resources to allocate from a user;
executing at least one process to compile said computer program, said executing further including for each process:

delaying executing said process if said maximum number of resources to allocate has been exceeded by a number of active processes;

dynamically adjusting said maximum number of resources to allocate if said resources are overloaded; and

executing said process.

36. A program storage device readable by a machine, tangibly embodying a program of instructions executable by the machine to perform a method for executing a distributed make command having a maximum number of resources to allocate as a parameter, the method comprising:

executing at least one process specified in a make file to compile said computer program, said executing further including for each process:

delaying executing said process if said maximum number of resources to allocate has been exceeded by a number of active processes;

dynamically adjusting said maximum number of resources to allocate if said resources are overloaded; and executing said process.

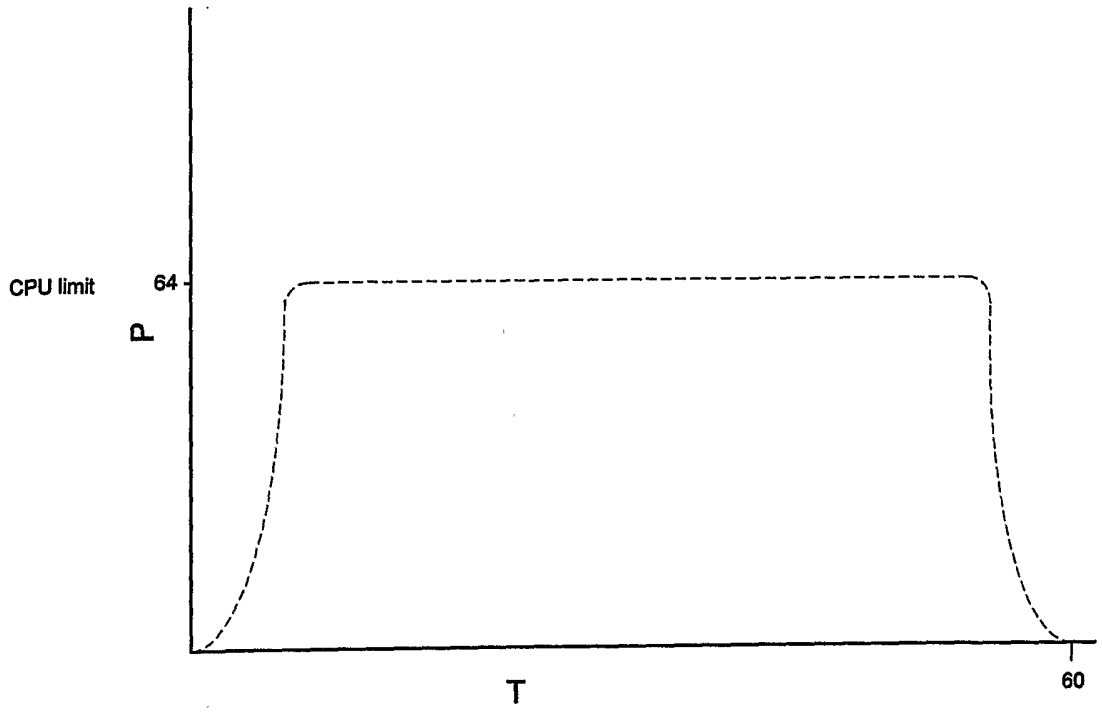
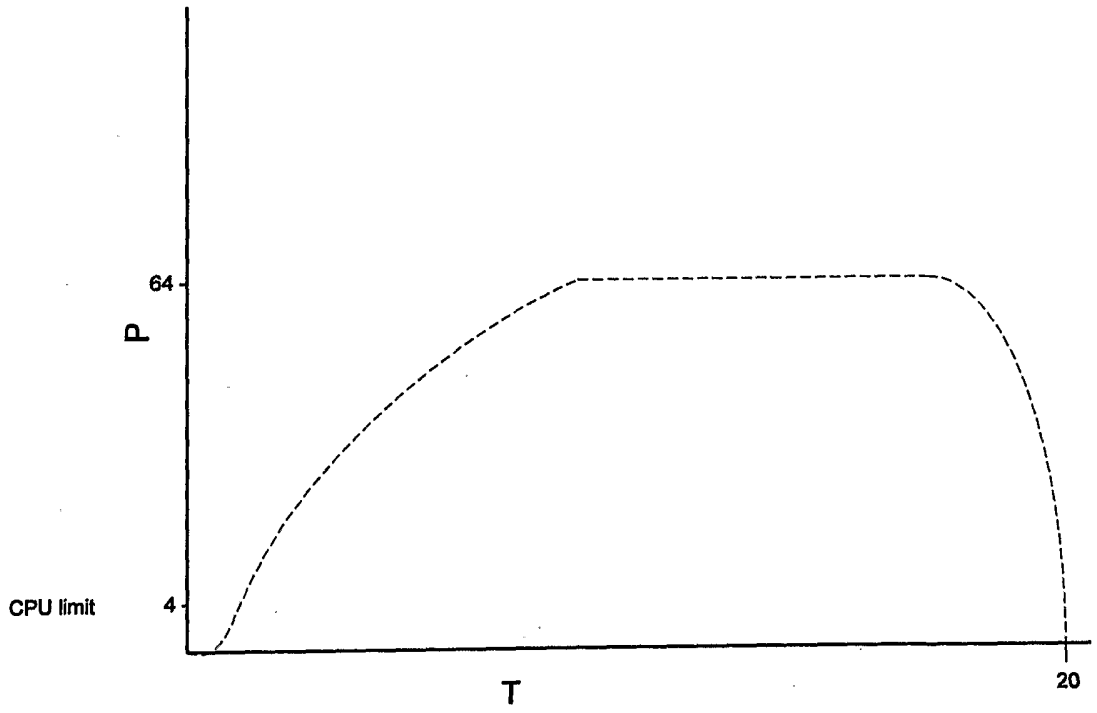


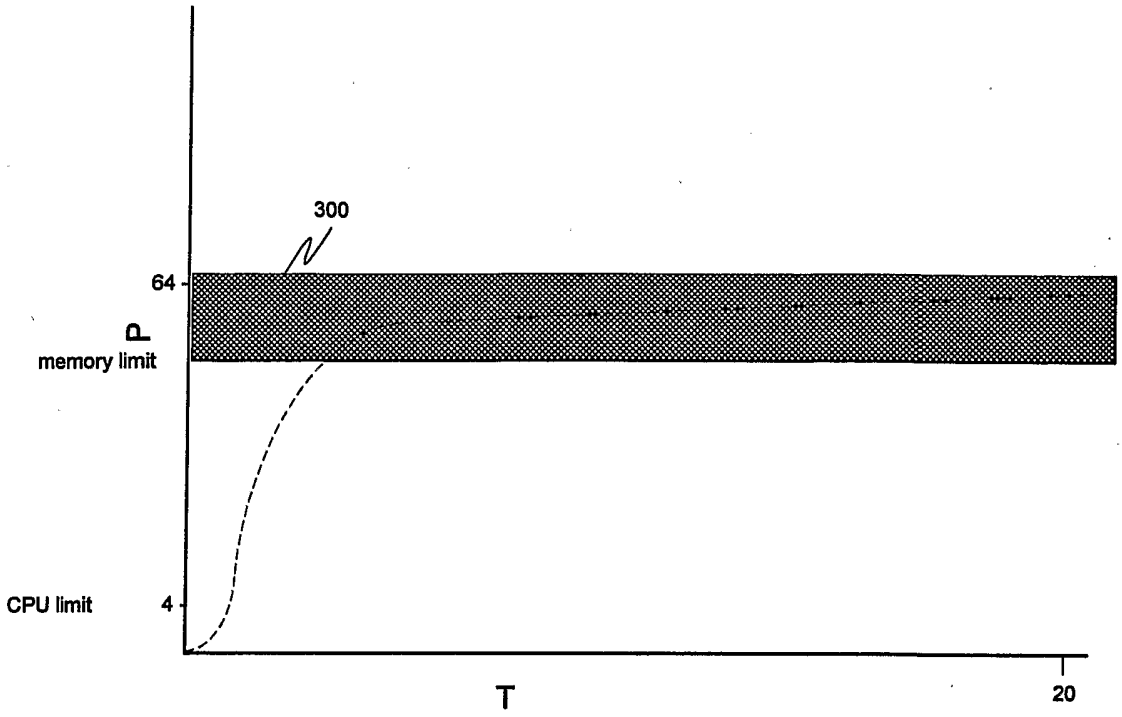
FIG. 1



T - time (minutes)

P - total number of processes

FIG. 2



T - time (minutes)

P - total number of processes

FIG. 3

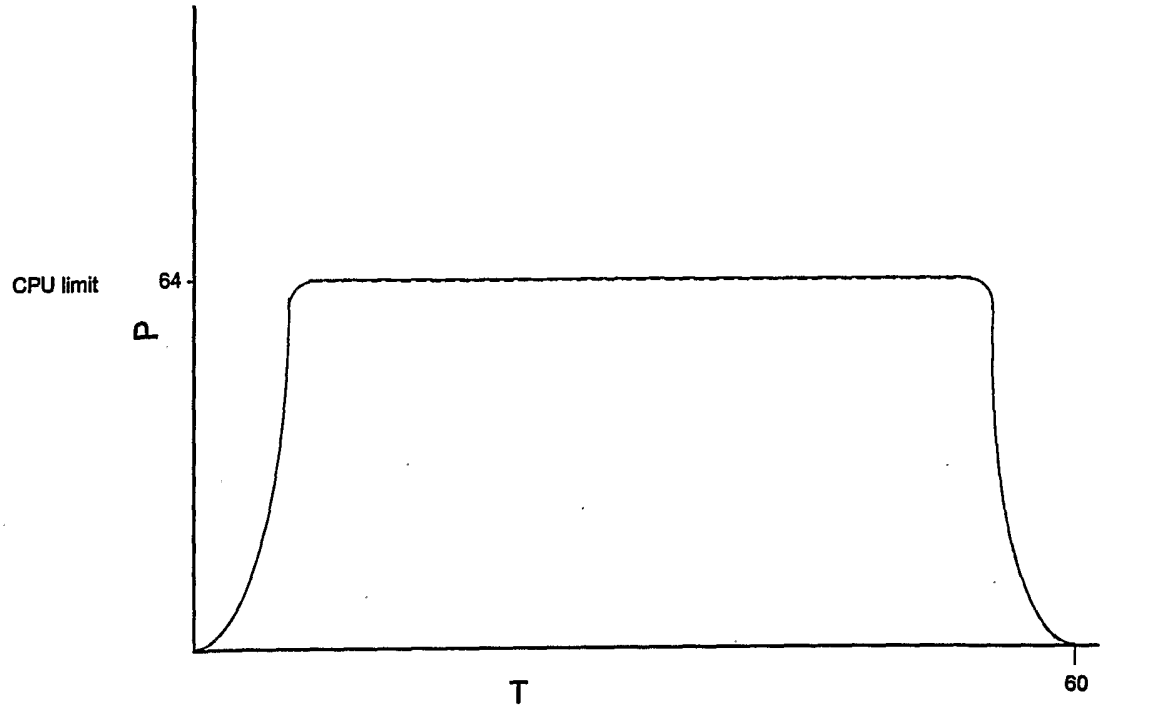

CPU limit
64
P
T
60
T - time (seconds)
P - total number of processes
Prior embodiment
Current embodiment

FIG. 4

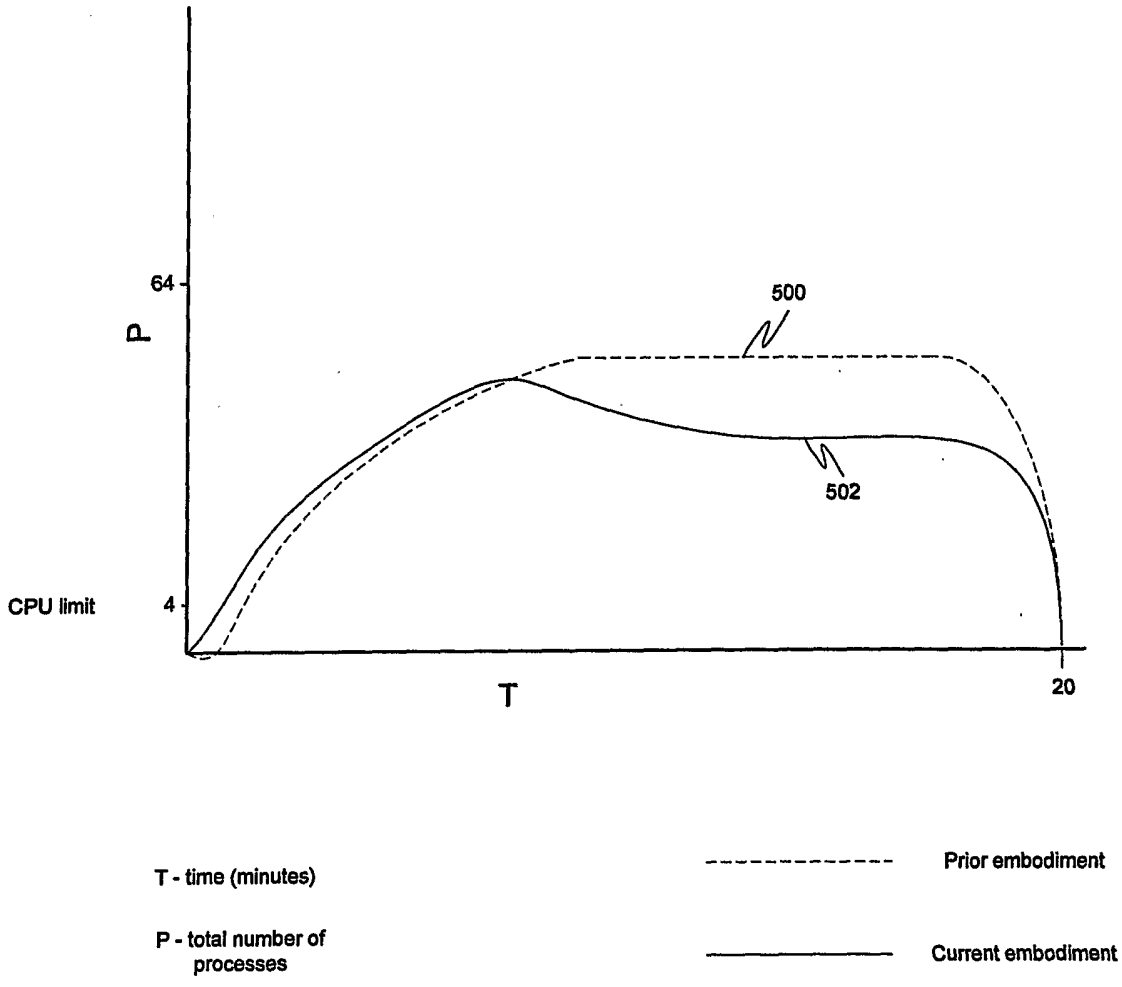

64
P
500
502
CPU limit
4
T
20
T - time (minutes)
P - total number of processes
Prior embodiment
Current embodiment

FIG. 5

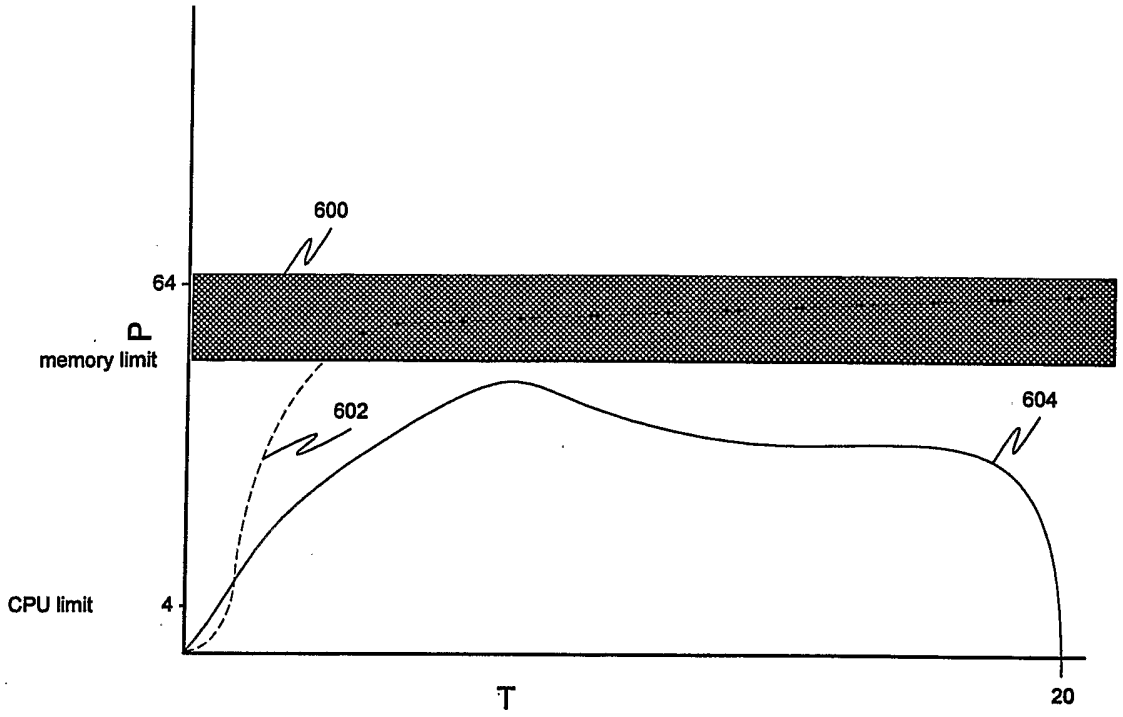

600
64
P
memory limit
602
604
CPU limit
4
T
20
T - time (minutes)
P - total number of processes
Prior embodiment
Current embodiment

FIG. 6

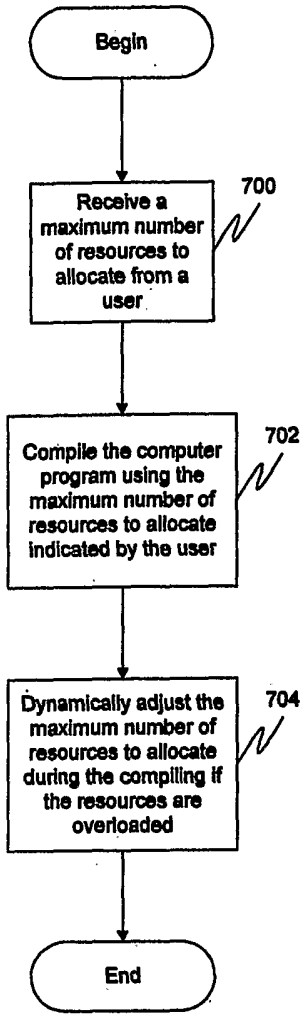

Begin
Receive a maximum number of resources to allocate from a user
700
Compile the computer program using the maximum number of resources to allocate indicated by the user
702
Dynamically adjust the maximum number of resources to allocate during the compiling if the resources are overloaded
704
End

FIG. 7

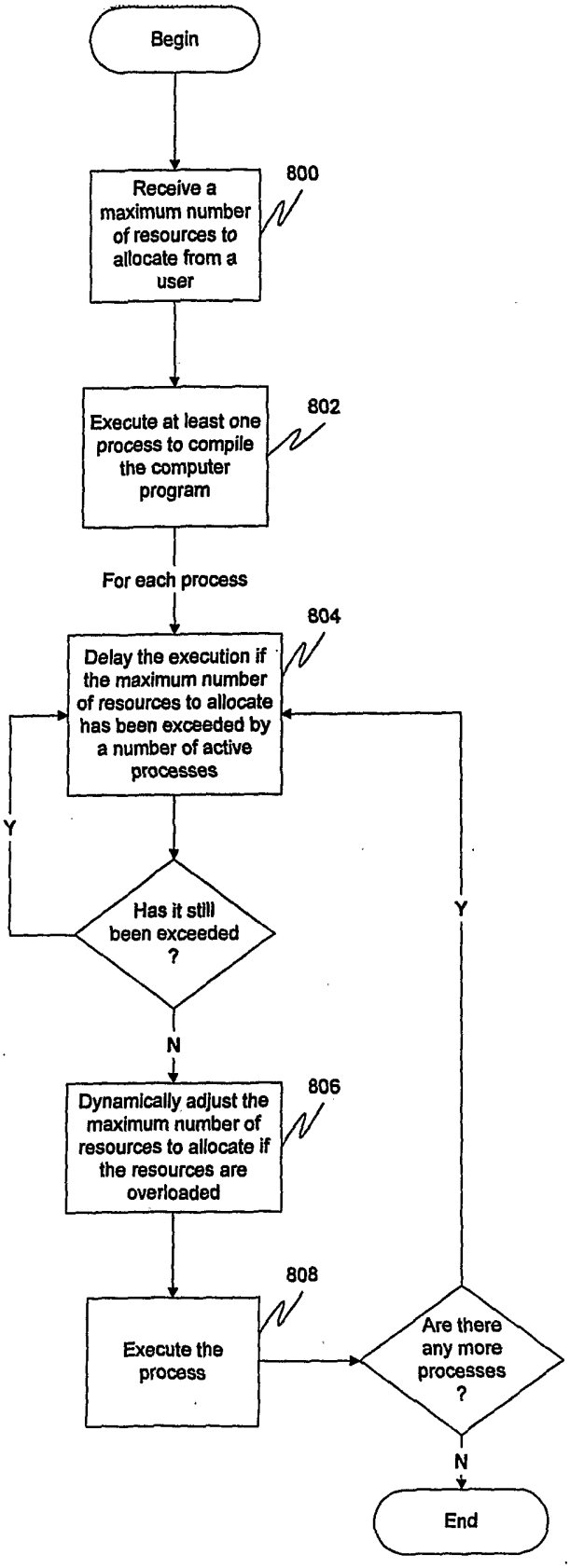

Begin
Receive a maximum number of resources to allocate from a user
800
Execute at least one process to compile the computer program
802
For each process
Delay the execution if the maximum number of resources to allocate has been exceeded by a number of active processes
804
Y
Has it still been exceeded ?
N
Dynamically adjust the maximum number of resources to allocate if the resources are overloaded
806
Execute the process
808
Are there any more processes ?
Y
N
End

FIG. 8

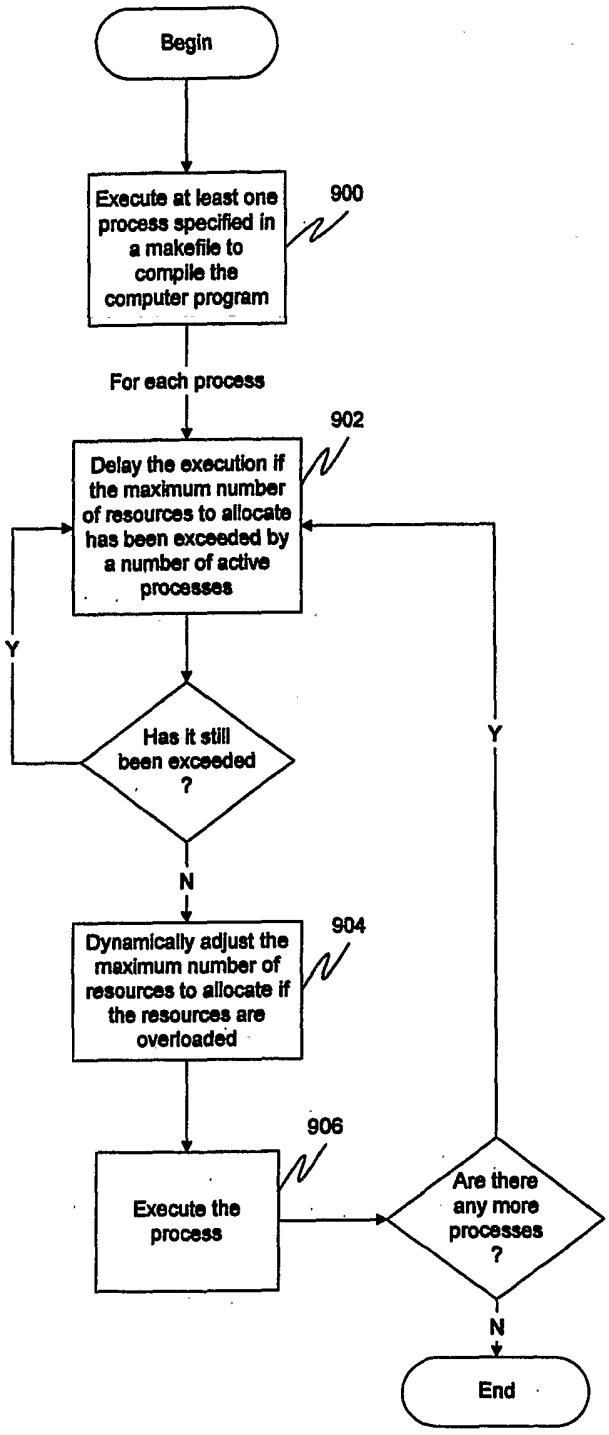

Begin
Execute at least one process specified in a makefile to compile the computer program
900
For each process
902
Delay the execution if the maximum number of resources to allocate has been exceeded by a number of active processes
Y
Has it still been exceeded ?
N
Dynamically adjust the maximum number of resources to allocate if the resources are overloaded
904
906
Execute the process
Are there any more processes ?
Y
N
End

FIG. 9

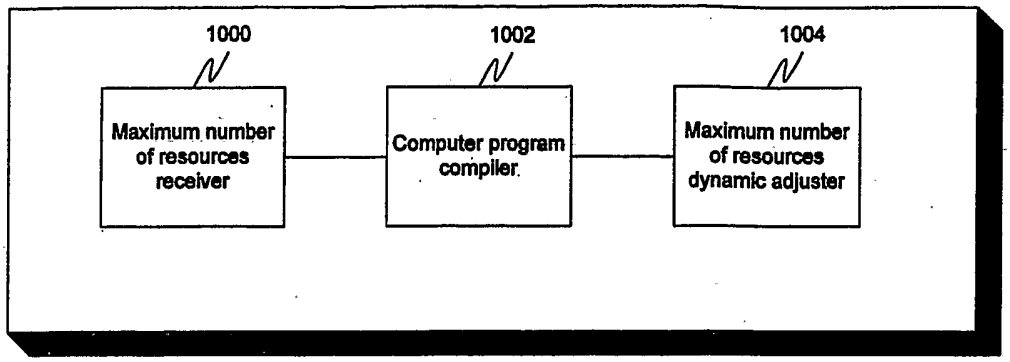

1000
1002
1004
Maximum number of resources receiver
Computer program compiler
Maximum number of resources dynamic adjuster

FIG. 10

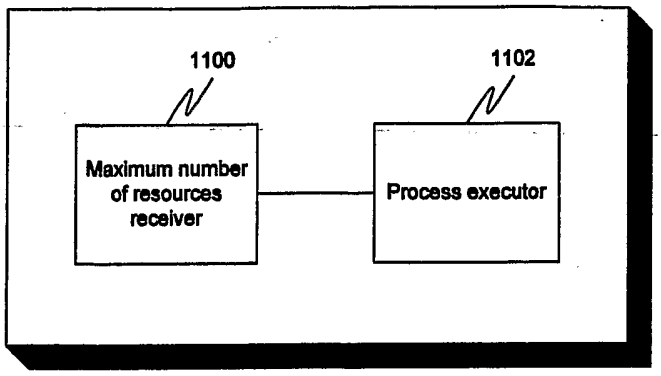

1100
1102
Maximum number of resources receiver
Process executor

FIG. 11

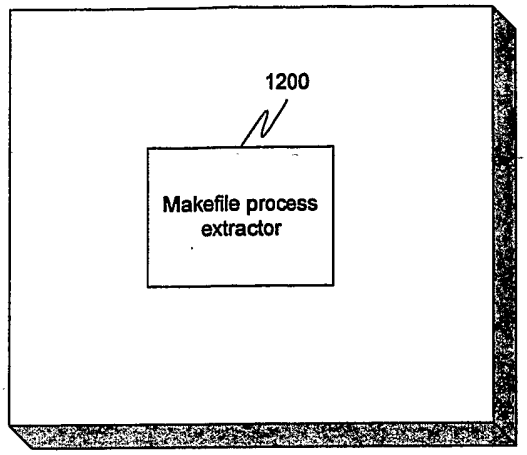


FIG. 12