Resource Management Methods for General Purpose Massively Parallel OS *SSS–CORE*

Yojiro Nobukuni, Takashi Matsumoto, Kei Hiraki

Department of Information Science, Faculty of Science, University of Tokyo
7-3-1 Hongo, Bunkyo-Ku, Tokyo 113, Japan

Abstract. We propose two resource management methods; a scheduling policy that reflects resource consumption states and a memory-replacement strategy based on page classification under distributed shared memory architecture. The performances of the two mechanisms are evaluated by a probabilistic simulation. An instruction-level simulator simulates variety of process sets with finite resources on proposed resource-management methods. The results show the superiority of the proposed resource management mechanisms.

1 Introduction

NUMA architecture [10, 1] is widely accepted as basic architecture for very high-performance computers because huge systems can be built by simply connecting many pairs of processing elements and local memories.

Current parallelizing methods for on NUMA systems [11, 2, 8] and optimization techniques for parallel applications[12, 7] optimize execution of a single parallel application on a fixed system configuration. Therefore, running multiple parallel applications with competing resource allocation in general-purpose environment is much less efficient than that is expected. Dynamic partitioning of parallel system is a necessary feature of a general-purpose parallel systems but it may further reduce the performance. Operating system level optimization is required to coordinate resources to run each process efficiently.

We propose two resource management mechanisms for efficient running of multiple parallel processes. One is a process scheduling mechanism that utilizes information on physical page usage. The other is a memory replacement mechanism based on page attribute used for distributed-shared-memory management. The performances of these mechanisms are evaluated by detailed probabilistic simulation.

Previous operating systems[5, 9] that allowed gang scheduling with dynamic repartitioning did not use resource informations and has limited scheduling flexibility. DHC[4] designed for UMA uses management structure close to ours but only uses load informations[3].

Section2 describes the resource management mechanisms. Section3 shows outline of an operating system, $SSS-CORE$ and proposes scheduling polisies

and page replacement policies. The methodology and the results of the simulation are given in section4 and 5 respectively. We conclude in section6.

2 Resource Management Mechanism

2.1 Scheduling Policy

Resource Management Tree To take resource consumption state into scheduling policy, resource related information must be managed. Our approach is to construct a data structure called *resource management tree* (RMT) to maintain system-wide resource usage and each process's resource consumption state.

RMT is hierarchically structured to be scalable. In addition, variants of parallel systems can be supported by adopting real structure of RMT to each specific systems, from workstation clusters to parallel super computers with flat networks. Scheduling decisions based on the structure naturally reflects the distances and hence the access costs between distributed resources. RMT has further advantages. It reduces the quantity of required physical resource for storing resource information. Bottlenecks of accessing the information is avoided.

Each node of the resource management tree logically holds information for resources seen below the node. They are number of processors and physical pages, number of total free processors and physical pages, and number of processors, physical pages and ID of each process. The root node additionally has priority, scheduling constraints and home node of each process. Figure 1 shows an example image for a four processor system with RMT.

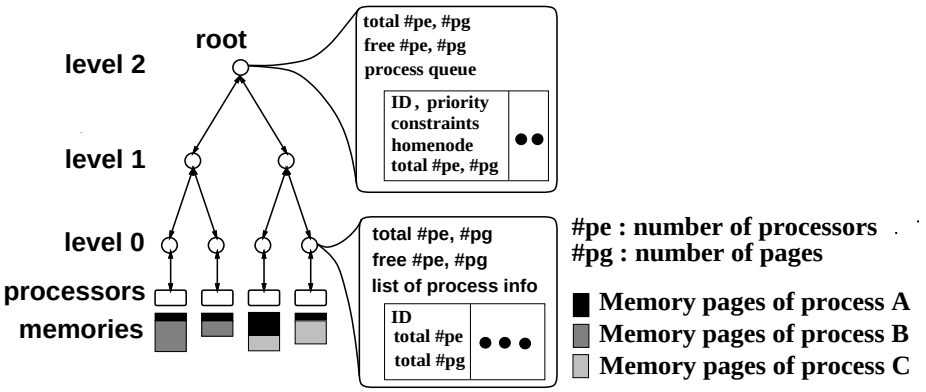

Fig. 1. Resource Management Tree

A process can gain performance by freely using allocated resources and by making the use of application level optimization. This can be done by using 2-level scheduling. The kernel allocates resources to each process by looking into the *resource management tree*. This way, resources that most fit for the use of a process can be allocated. The resource allocation within a process is left to user level scheduler, which freely re-allocate resources into internal threads.

Scheduling Constraints A parallel optimizing compiler[12] generally assumes that system resources are used by a single application. Our goal is to run object codes efficiently in a multiple user/multiple process environment. In such an environment, a process can achieve higher speedup when allocated resources satisfy its requirements and preferences. Scheduling constraints are used by processes to specify these informations to the kernel. The kernel follows the given constraints so that the requirements of a process can be satisfied as much as possible.

A process can use scheduling constraints to specify its requirements of and preferences for the number of processors to use, communication cost between processors, memory access costs, and process migration. The *fixed processors constraint* expresses a constant number of processors a process requires. The *variable processors constraint* enables a process to be allocated variable number of processors.

Priority Computation Since resources are allocated to satisfy each process requirements, a mechanism must be arranged to coordinate fair sharing of resources. Fairness can be achieved by managing priorities according to the amount of used resources and strength of given scheduling constraints and scheduling in priority order. Aging priorities according to these terms realizes fairness.

Priority is based on following values; (1)amount of used resources: U_r, (2)strength of scheduling constraints: R_c, (3)degree of constraints satisfaction: $S_c = 0$ *or* 1, (4)amount of wasted resources: W_r, (5)presence of waiting process: $f_w = 0$ *or* 1 The aging value of a process is computed from next expression.

$$aging = (U_r R_c S_c + W_r) f_w * t - C_r(1 - t)$$

Smaller the value, higher the priority. C_r is the aging coefficient. To prevent priority values to divert, the sum of the aging values of processes that were running before the time slice is equally divided and distributed to waiting processes.

2.2 Memory Replacement Strategy

Under general environment where multiple processes execute simultaneously, system must be designed to bear situations when physical memories are exhausted. Selecting victim pages from those that are less frequently accessed and have less re-reference cost will enhance system performance.

Pages that belong to the currently running process are usually more referenced. Generally, local pages are more referenced than those shared by threads. Suppose a shared copy page has been replaced, it can be obtained with small cost through network from remote cluster. However, replacing pages that invoke disk accesses on re-reference can be a source of slow down.

3 SSS–CORE

The mechanisms proposed in the paper are planned to be built into an operating system called *SSS–CORE*. The performance of the resource management mechanisms introduced are evaluated by simulating system with *SSS–CORE*.

SSS–CORE[6] is a general purpose massively parallel operating system for NUMA parallel distributed systems. It provides multiple user/multiple job environment with timesharing and space partitioning. The main objective of *SSS–CORE*is achieving maximum performance from each parallel application.

SSS–CORE provides a mechanism that allows information transfer between kernel and user level as described in the previous section. The information includes those used by the resource management mechanisms, e.g. usage of physical pages and number of processes on each node.

3.1 Scheduling Policy

The performances of five kernel-level scheduling policies are compared through evaluation. Every policy computes process priorities according to resource consumption state at each time slice and schedules processes with highest priorities. Processors are looked for within a particular area and allocated to a process if sufficient number of processors are found in the are. The policies are described below.

Policy0 allocates randomly selected requested number of clusters
Policy1 allocates requested number of continuous clusters in a fixed order
Policy2 first allocate clusters in home-node area where pages of target process exist, then clusters in whole area will be tried on failure.
Policy3 same as **Policy2**, but only home-node area is tried
Policy4 same as **Policy3**, but clusters that actually has target process's pages are allocated

The home-node of a process represents a subtree of the resource management tree that includes its requested number of processors. It somewhat corresponds to the area where the process was previously scheduled. Figure 2 gives a home-node example. Process A in the figure, which has pages at marked clusters in area4, takes a node as its home-node which represents the subtree in area3. Area3 is called *home-node area* of process A.

Policy2, **3**, **4** use resource management tree. The difference among these three policies is in how much they persist in allocating processors from clusters where process's currently using physical pages are located. The difference is in the action taken when sufficient number of processors cannot be prepared by those clusters. **Policy2** looks for processors for all clusters. **Policy3** tries to allocate from clusters in home-node area; the subtree below the home-node of target process. **Policy4** gives up scheduling the target process. Figure 2 shows the area where each of policies look for processors to allocate. Area 2, 3, 4 corresponds to the area for **Policy2**, **3**, **4** respectively. **Policy4** mostly schedules a process

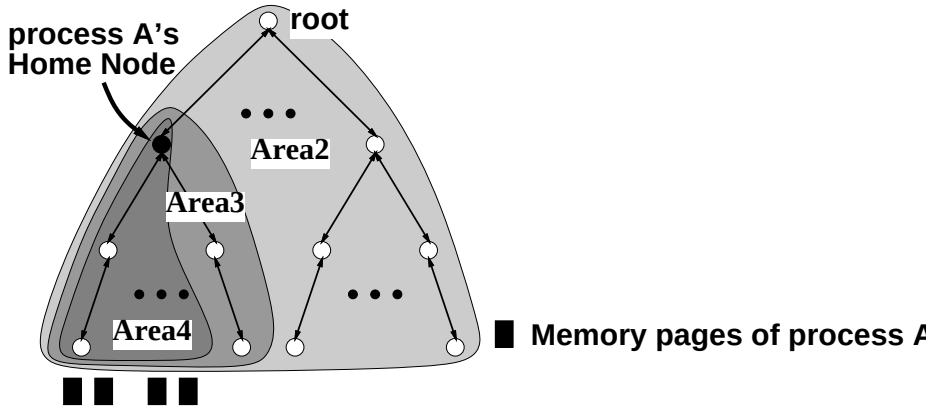

Fig. 2. Scheduling Target Area

to the same processors time to time. Chances that processes will be scheduled to clusters where they hold physical pages are greater in **Policy4**, **3**, **2** order. More processors may be utilized in reverse order.

Defining as many user-level schedulers as the number of processes to simulate is not possible. A single policy is defined and used by all processes. It schedules the identical threads to the processors that were also allocated to the process at previous allocation by the kernel level scheduler.

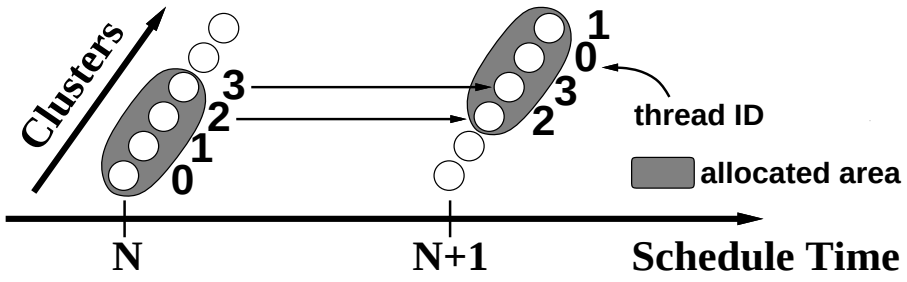

Fig. 3. An Example of User Level Scheduling

Figure 3 shows an example of thread scheduling. In the example, processors that are allocated to the process by $(N+1)$st scheduling as well as (N)th scheduling will run threads 3 again. Processors that are newly allocated to the process at $(N+1)$st scheduling will run remaining threads (threads 0, 1, 2) in thread ID order. When thread is scheduled to different clusters, its local pages must be transferred through the network. Distributed shared memory system is responsible for properly transferring the shared pages of the thread. Clearly, the more overlaps in allocation area, the lesser the amount of page transfers.

Note that when time quantum is sufficiently larger than the time required for context switching, required time for computing scheduling itself is relatively small. *SSS–CORE* will use larger value for time quantum. The time required for

scheduling is ignored in the simulation.

3.2 Page Replacement Strategies

Two page replacement strategy is evaluated and compared with each other.

Strategy0 Simple LRU without page classification
Strategy1 Uses page class. Processes are scanned in reverse priority order.

Assuming distributed shared memory system, memory pages can be classified into 6 groups by pointing whether a page; (a) belongs to currently running process or not, (b) is shared page or local page, and (c) has other copy pages or not. The page classes for **Strategy1** are; (1) copy page of not running process, (2) copy page of running process, (3) last one page of not running process, (4) local page of not running process, (5) last one page of running process, and (6) local page of running process. "Last-one" page in the list means a shared page without any copy thus requires a disk access on next access. Ordering between classes 4 and 5 cannot be given trivially. Class 4 is prior to 5 in the list to maximize the efficiency of currently running process.

Since processes are scheduled in priority order, pages of lower priority process are possibly less referenced. **Strategy1** utilizes this characteristic. Both strategies will not select coherency processing shared pages as the victim page for replacement.

Table 1. Process set for simulation

Process set	Number of proc's	Parallelism (Processes with the same parallelism)	Total parallelism	local mem size [pages]	shared mem size [pages]	VR ratio [VR ratio]	Sync Interval [clocks]
A	12	16,36,48,50(2),64,70 96,100,128,192,200	1050	35500	10120	1.00	10000–20000
K	12	48(5),96(5),208,256	1184	49920	6360	1.30	10000
L	11	64(5),128(2),192(2),256(2)	1472	68480	7280	1.68	10000
M	23	16(16),50(6),256(1)	812	35480	30320	1.70	10000

Table 2. Parameters and Costs

Parameters	Values
Number of processors	256
Average memory access interval	10 clk
Disk access cost	100000 clk
Page transfer startup cost	500 clk
Communication startup cost	50 clk
Pages per cluster	400 pages
Page size	4096 Byte
Total memory	409.6 Mbyte
1 quantum	1000000 clk

4 Simulation Model

Operating system level simulation of parallel architecture by highly detailed model is practically impossible. Executing a particular suite of applications on the simulator is not sufficient for evaluating a general purpose operating environment. Therefore, even instruction level simulation does not fit for the objective. We use a detailed probabilistic model for simulation. Probabilistic model is good for simulating variety of processes. However, a stream of process activities cannot easily be given the meaning from program point of view.

A pair of a processor and a memory constructs a cluster. Clusters are connected by tree structured doubly linked interconnecting network. The value a message actually takes for moving one-hop is computed from basic transfer cost of each type of messages and the bandwidth of the network connection where it is passing.

A process has as many number of threads as the number of processors it requests. It uses the *fixed number of processors scheduling constraint* , and thus its parallelism never changes through its life time. Threads here denotes the execution context of a parallel process at a cluster. A thread of a process has own local memory space and a shared memory space shared among threads of the process. Both memory spaces are provided with reference frequency tables that describe how frequently each page of the space is accessed.

Pages of shared space are managed by distributed shared memory system. Sequential consistency memory model with an update protocol is used. Every write access starts update processing by sending update messages to every copy. The processor stops until it collects all acknowledges. To model NUMA systems, memory access cost and basic communication costs are set as to satisfy "local access $\ll$ inter-cluster access$\ll$ disk access". When threads change clusters on which it executes, its local pages are moved on-demand through network. Shared copy pages that do not reside on currently allocated clusters are removed without any cost. Accesses to unloaded virtual pages will cause disk accesses.

Process execution is clock-based probabilistic model. Processes make memory reference actions at each clock if possible. With given interval of effective execution clocks, randomly selected threads of a process synchronize by a simple barrier. Effective execution means the time or clocks spent for other than waiting for synchronization to complete or for memory access processing to end.

5 Simulation Results

The parameters used in simulation are shown in Table 2. System with as many as 256 processors is evaluated. The topology of the network is three and four leveled tree structure. The former expands at root level into 4-way, then 8-way, and 8-way at the bottom level (**w488**). The latter expands 4-way at each level of the network(**w4444**). Table 1 describes the sets of parallel processes simulated.

Each experiment is carried on until one of the processes in a set stops execution or 100 time slices has passed. The results are shown in Figure 6, 4, 6 and

6. Graphs on the left columns are results of **w488** system, and on the right are of **w4444** system.

Each group in a graph is the results of scheduling **Policy0** through **Policy4**. Three left bars of a group are the results of **Strategy0** and the others are of **Strategy1**. Three evaluated values are plotted; (1)net effective execution rate, (2)calibrated effective execution rate. (3)maximum effective execution rate, and (1) is total efficiency of processors when processes are scheduled. Idle processors to which no processes are scheduled are not included. All processor idle times are accumulated to (3) and (2) is computed by following expression, (1) * (1.0 + idle time rate). Processor idle times are accumulated with the ratio of net effective execution rate.

5.1 Kernel Level Scheduling

Policy4 shows the best performance even when compared by net effective execution rate, which is disadvantageous for **Policy4** because processor idle times are not included. When compared by other evaluations, the difference becomes larger. On real systems, processor idle time can be reduced by following two methods.

1. using *variable number of processor scheduling constraint* (will be introduced to *SSS–CORE*), processors can be flexibly utilized for number of processors.
2. un-allocated spaces caused by scheduling oriented processor fragmentation can be utilized by another process not included in a particular set of processes.

In case 1, the calibrated performance can generally be expected because processes will utilize the newly allocated processor space by *variable number of processor scheduling constraint* as much efficiently as they used the same space when allocated by *fixed number of processors scheduling constraint*. When an additional process is assumed for a particular set of processes, it can use the processors in formerly fragmented space as much efficiently as maximum effective execution rate, depending on its characteristics as a parallel process. Thus maximum effective execution rate can be expected for case 2. Evaluating cases for variant process sets other than those experimented is inevitable and important for describing the performance of general purpose operating system and prospecting how the performance of *SSS–CORE* will be. Comparison by calibrated or maximum effective execution rate is validated from this point of view.

Policy4 is the best in efficiency by any of the three estimations. The quantity of page transfer is larger among **Policy2**, **3**, **4** in the order. Table 3 shows that the time spent in synchronization or communication get larger for policies in the same order. Changing processor allocation space time to time cause each memory to be filled with pages from many processes (Disk acssessing time is not included). When **Policy2** or **Policy3** is used, processes scramble for the physical pages and result in lower in efficiency than **Policy4**.

5.2 Memory Replacement Strategies

Strategy1 always outperforms **Strategy0**. Table 4 is the breakdown of replaced counts for each class of pages. Results of the scheduling **Policy4** on **w488** system is shown.

As for **Strategy1**, mostly copy pages are replaced. Process sets A and K, which impose small physical memory requirement, see only copy pages of not running processes victimized. But for **Strategy0**, local pages and last-one shared pages are replaced.

The results show that selecting victim pages according to the classification enhances system performance. Even when the system is somewhat highly loaded, efficiency is preserved by not kicking local pages that are more frequently referenced out of memory.

5.3 Considerations on Realizability of General Environment

Maximum efficiencies of the results are roughly between 65% to 85% for the experiment of scheduling policy **Policy4** and **Strategy1** replacement strategy pair. The lower results come from sequential consistency memory model. The time waiting for preceding accesses to complete is very large. Update processing time is not very large compared to the waiting time. Cooperating more relaxed memory model and lighter consistency managing system solves the problem. In addition, performances of parallel applications with large shared access frequencies can be enhanced with various compilation techniques and by introducing useful communication techniques, such as hierarchical multicasting and acknowledge combining. Thus, lower simulation results does not negate general environment on parallel distributed system.

6 Conclusion

The paper has described kernel level scheduling policy that uses information of resource consumption state and memory replacement strategy that uses page

Table 3. Execution Time Breakdown (%) (w488, Strategy1 replacement)

Set	Type	algo0	algo1	algo2	algo3	algo4
A	Max	57.89	57.36	60.41	64.22	84.02
	Sync	19.98	19.59	20.84	20.03	12.45
	Comm	22.10	23.01	18.72	15.70	3.49
K	Max	42.65	56.57	59.53	59.53	74.91
	Sync	25.87	22.08	23.24	23.24	17.82
	Comm	31.46	21.32	17.20	17.20	7.22
L	Max	38.81	48.93	54.98	54.98	65.85
	Sync	28.52	24.07	23.58	23.58	22.23
	Comm	32.64	26.96	21.39	21.39	11.87
M	Max	24.79	35.73	47.42	61.47	64.37
	Sync	22.83	19.03	18.68	16.57	16.06
	Comm	52.36	45.21	33.86	21.90	19.51

Table 4. Replaced times for each class of pages(**Policy4**, **w488**)

Page Replace Strategy	Proc Sets	Page Class					
		other's copy	own copy	other's last	other's local	own last	own local
Strategy 0	A	10944	4770	723	4763	303	1130
	K	33105	10487	830	22598	65	3803
	L	92253	18063	4911	63885	136	9978
	M	46394	23581	6873	11725	4054	6577
Strategy 1	A	38976	0	0	0	0	0
	K	77665	0	0	0	0	0
	L	414287	0	0	0	0	0
	M	245705	11578	0	0	0	0

classification upon distributed shared memory system. The performances of various methods for these mechanisms are evaluated by simulating on detailed probabilistic model.

As for the kernel level scheduling, the simulation results showes the superiority of those policies that use resource management data structure and allocate processors of clusters with memory affinity to a process. Replacing pages according to the page classification is found much superior than replacement policy by simple LRU order without the classification. When these two mechanisms are in-cooperated together, the effective execution rate of the system is higher than 65% for highly loaded cases.

Acknowledgement

The work is supported by IPA Advanced Information Technology Program (AITP) of Information-technology Promotion Agency (IPA), Japan.

References

1. Intel Supercomputer Systems Division. *Paragon User's Guide*, order number 312489-003 edition, June 1994.
2. F. Douglis and J. K. Ousterhout. Process Migration in Sprite Operating System. *Proc. of the 7th Inter. Conf. on Distributed Computer Systems*, September 1987.
3. D. G. Feitelson. Packing schemes for gang scheduling. In *Proc. IPPS'96 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 54–66, April 1996.
4. Dror G.Feitelson and Larry Rudolph. Distributed Hierarchical Control for Parallel Processing. *IEEE Computer*, 23(5):65–77, May 1990.
5. B. C. Gorda and E. D. Brooks III. Gang Scheduling a Parallel Machine. Technical Report UCRL-JC-107020, Lasrence Livermore NL, December 1991.
6. T. Matsumoto, S. Huruso, and K. Hiraki. General Purpose Massively Parallel Operating System SSS–CORE. *Proceedings of 11th Japan Society for Software Science and Technology*, pages 13–16, October 1994. (in Japanese).
7. Takashi Matsumoto. Synchronization mechanisms and processor scheduling on multiple processors. *IPS Japan SIG report*, pages 1–8, November 1989. (in Japanese).
8. Takashi Matsumoto. Elastic barrier: Generalized barrier synchronization mechanism. *Trans. of IPS Japan*, 32(7):886–896, July 1991. (in Japanese).
9. J. K. Ousterhout, D. A. Scelza, and P. S. Sindhu. Medusa: an experimant in distributed operating system structure. *Comm. ACM*, 23(2):92–105, February 1980.
10. J. Palmar and Jr. G. L. Steele. Connection Machine model CM-5 system overview. *4th Symp. on Frontiers Massively Parallel Comput.*, pages 474–483, October 1992.
11. T.E.Anderson et al. Scheduler activations: Effective kernel support for the user-level management of parallelism. *Proc. of the 13th ACM Sympo. on Operating Systems Principles*, 25(5):95–109, October 1991.
12. R. P. Wilson and et al. SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers. *ACM SIGPLAN Notices*, 29(12):31–37, December 1994.

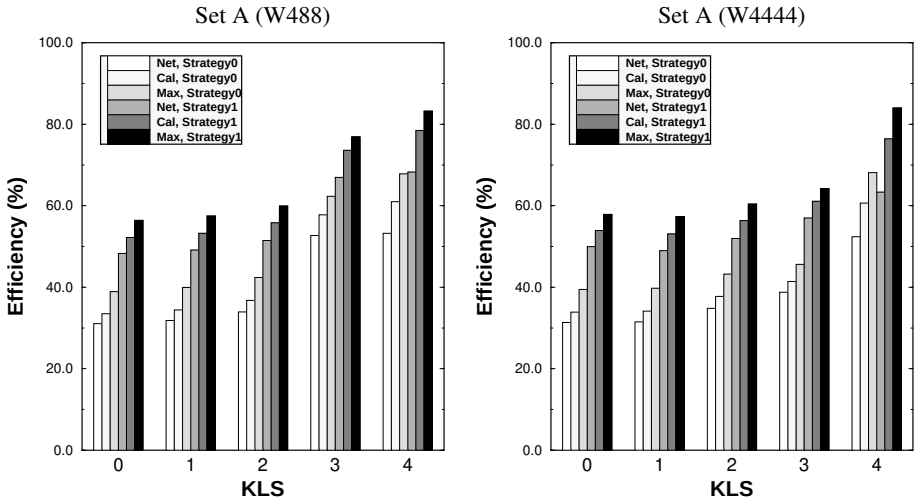

Fig. 4. Results of Process Sets A **w488**(Left), **w4444**(Right)

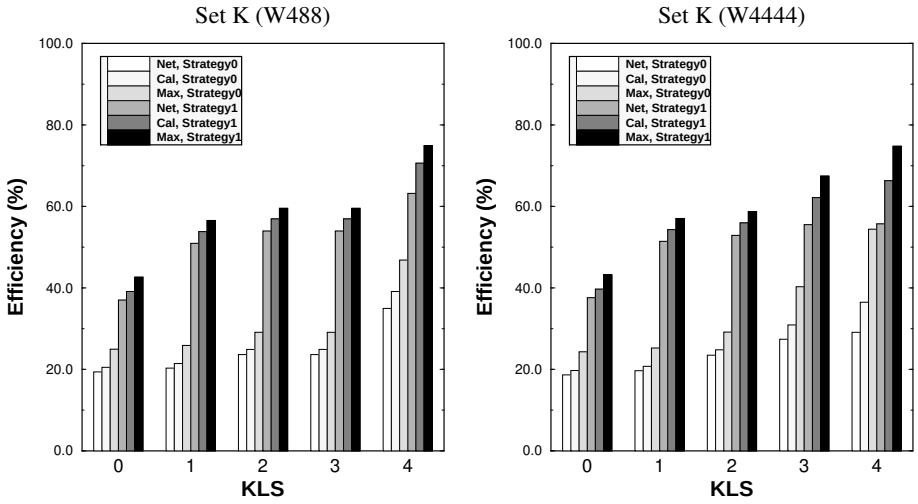

Fig. 5. Results of Process Sets K **w488**(Left), **w4444**(Right)

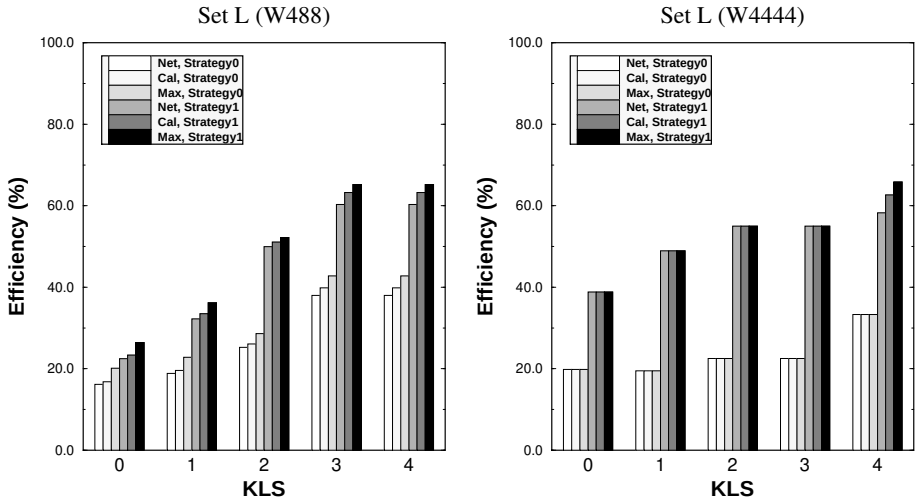

Fig. 6. Results of Process Sets L **w488**(Left), **w4444**(Right)

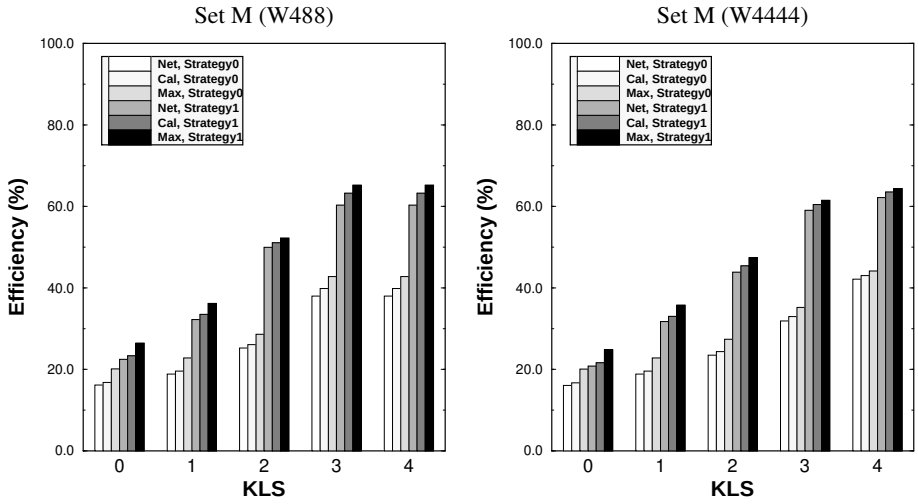

Fig. 7. Results of Process Sets M **w488**(Left), **w4444**(Right)