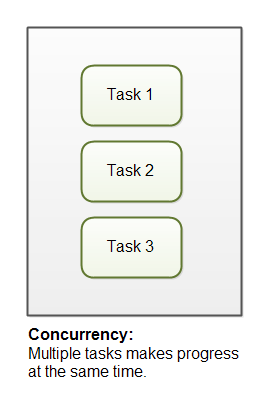




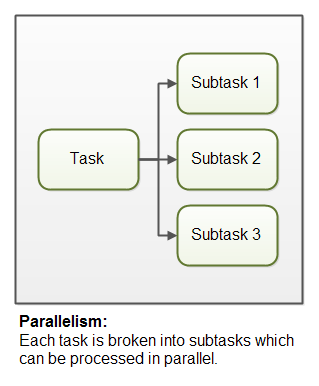
**Concurrency**

Concurrency means that an application is making progress on more than one task at the same time (concurrently). Well, if the computer only has one CPU the application may not make progress on more than one task at *exactly the same time*, but more than one task is being processed at a time inside the application. It does not completely finish one task before it begins the next.



**Parallelism**

Parallelism means that an application splits its tasks up into smaller subtasks which can be processed in parallel, for instance on multiple CPUs at the exact same time.



Concurrency is related to how an application handles multiple tasks it works on. An application may process one task at at time (sequentially) or work on multiple tasks at the same time (concurrently).

Parallelism on the other hand, is related to how an application handles each individual task. An application may process the task serially from start to end, or split the task up into subtasks which can be completed in parallel.

An application can be concurrent, but not parallel. This means that it processes more than one task at the same time, but the tasks are not broken down into subtasks.

An application can also be parallel but not concurrent. This means that the application only works on one task at a time, and this task is broken down into subtasks which can be processed in parallel.

Additionally, an application can be neither concurrent nor parallel. This means that it works on only one task at a time, and the task is never broken down into subtasks for parallel execution.

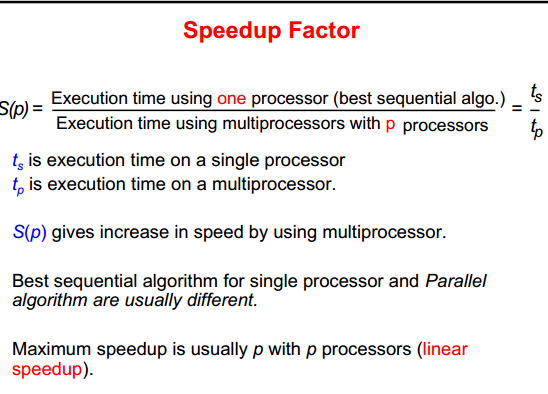
Finally, an application can also be both concurrent and parallel, in that it both works on multiple tasks at the same time, and also breaks each task down into subtasks for parallel execution. However, some of the benefits of concurrency and parallelism may be lost in this scenario, as the CPUs in the computer are already kept reasonably busy with either concurrency or parallelism alone. Combining it may lead to only a small performance gain or even performance loss. Make sure you analyze and measure before you adopt a concurrent parallel model blindly.

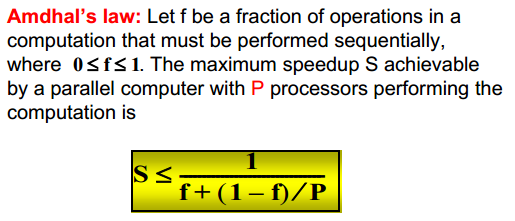
Concurrency can be achieved by multiprocessing and time-sharing

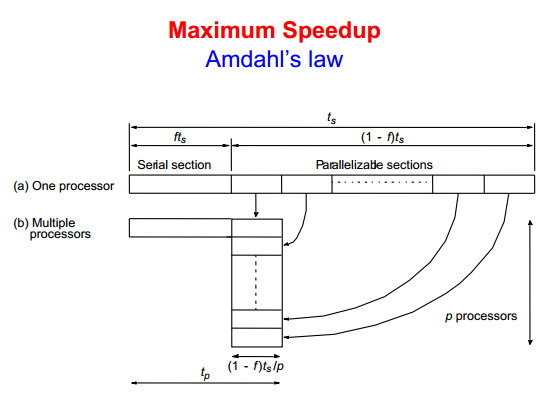
.

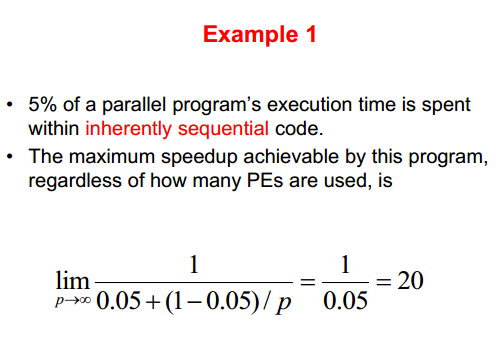
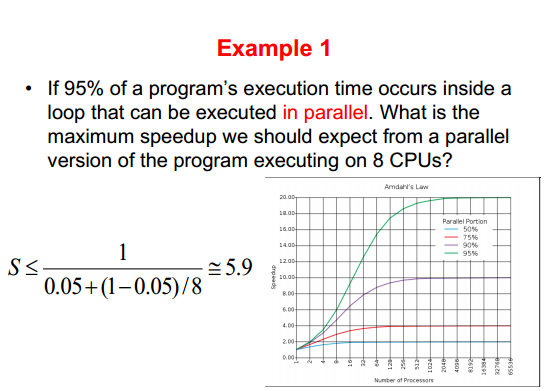


Speedup is the ratio between the time needed for the most efficient sequential algorithm to perform a computation and the time needed to perform the same computation on a machine incorporating data/control computation.

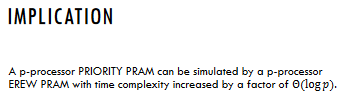
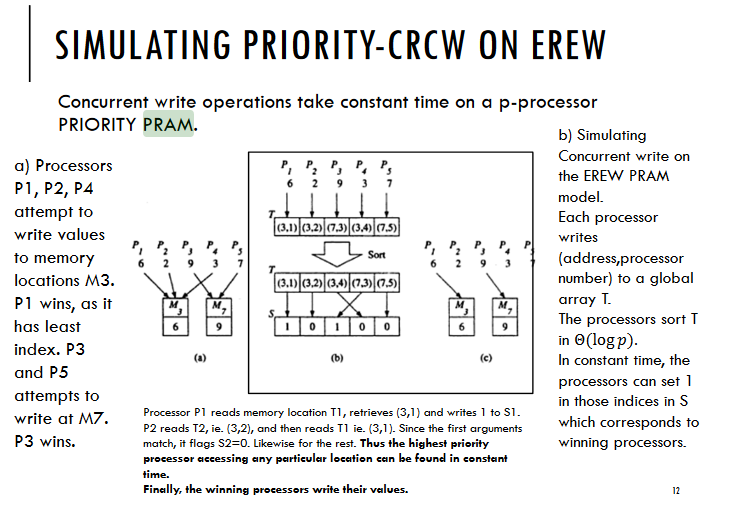




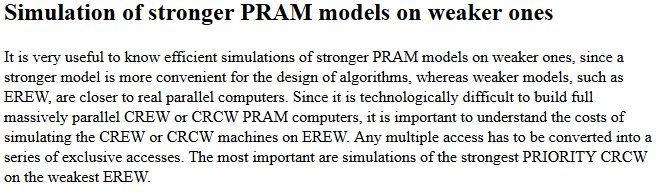


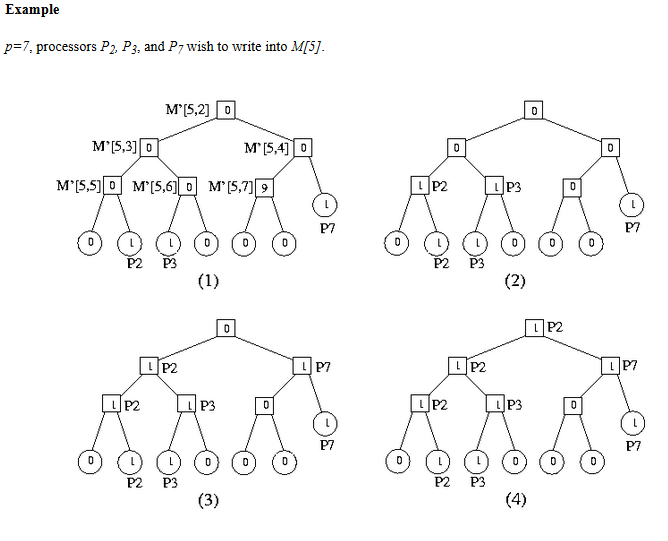
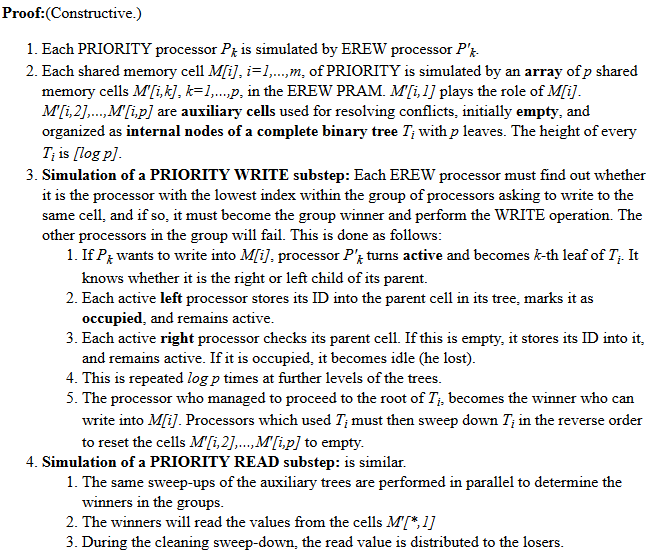
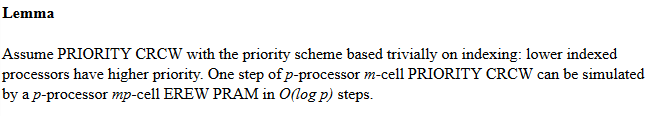




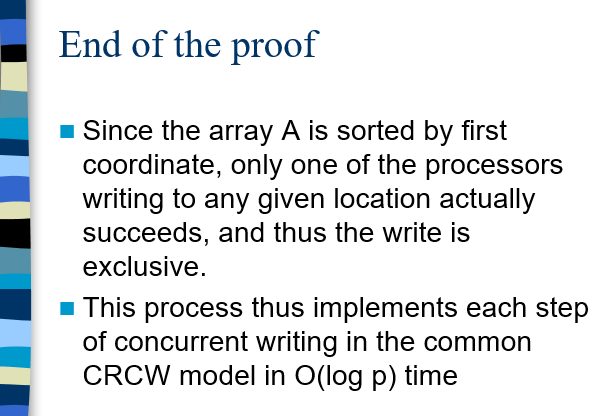
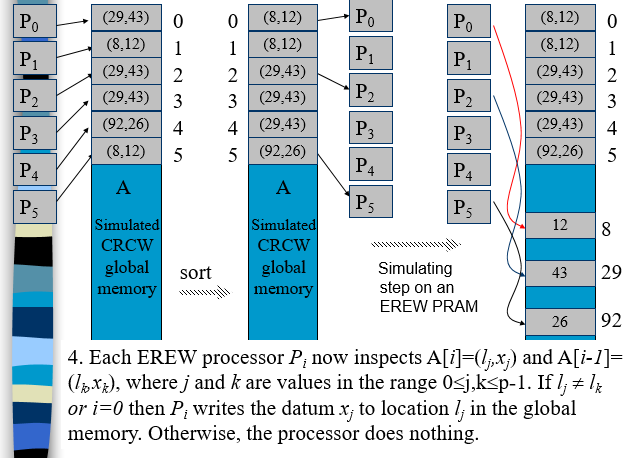
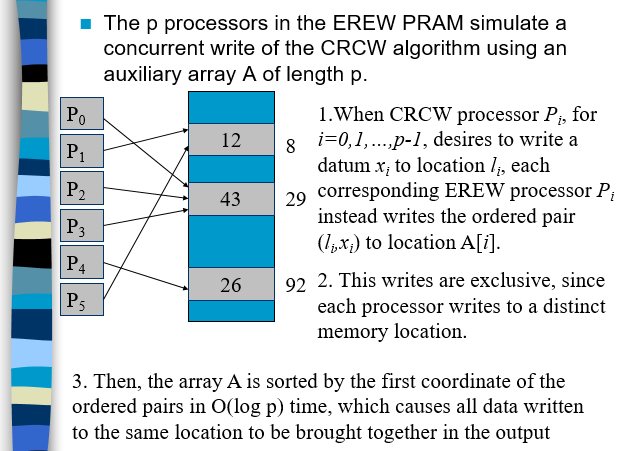
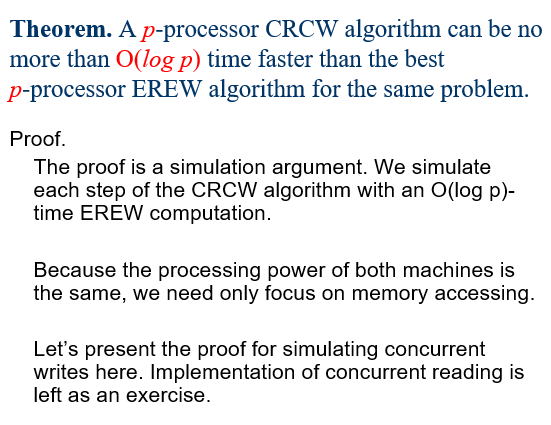


<http://cse.iitkgp.ac.in/~debdeep/courses_iitkgp/PAlgo/Autumn16-17/slides/Lect3PRAM.pdf>

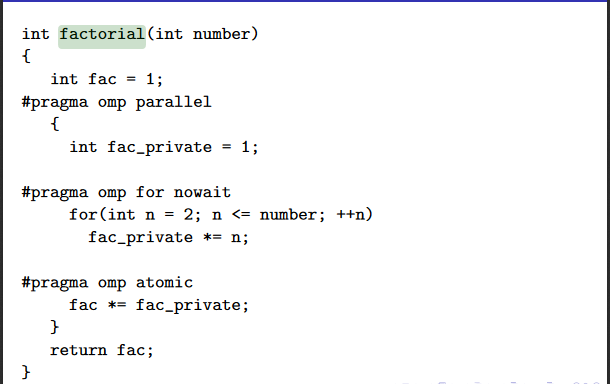




<http://pages.cs.wisc.edu/~tvrdik/2/html/Section2.html>







<https://fenix.tecnico.ulisboa.pt/downloadFile/3779578814853/cpd-08.pdf>

[About doubling technique](http://home.wlu.edu/~whaleyt/classes/parallel/topics/doubling/doubling.html)

**#include <stdio.h>**

**#include "mpi.h"**

**int** main(**int** argc, **char** \*argv[])**{**

**int** myRank;

**int** size;

**int** fact;

**int** lower,upper;

**int** i;

**double** local\_result = 1.0;

**double** total;

/\* initialize MPI \*/

MPI\_Init(&argc,&argv);

/\* get my rank and the size of the communicator \*/

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myRank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

/\* get the input. (only if i have rank 0) \*/

**if**(myRank==0)**{**

printf("Enter a number:");

scanf("%d",&fact);

**}**

/\* since only the process with rank 0 has the input,

\* we must pass it to all the other processes. \*/

MPI\_Bcast(&fact, /\* in/out parameter \*/

1, /\* count \*/

MPI\_INT, /\* datatype \*/

0, /\* root \*/

MPI\_COMM\_WORLD); /\* communicator \*/

/\* calculate the upper and lower boundaries

\* for each process

\*/

**if**(myRank==0)**{**

lower = 1;

**}else**

lower = myRank \* (fact / size) + 1;

**if**(myRank==(size-1))

upper = fact;

**else**

upper = (myRank + 1) \* (fact / size);

/\* now that we know upper and lower, do the

\* multiplication in our local area

\*/

**for**(i=lower;i<=upper;i++)**{**

local\_result = local\_result \* (**double**)i;

**}**

/\* combine all the local results by multiplying them

\* together

\*/

MPI\_Reduce(&local\_result, /\* operand \*/

&total, /\* result \*/

1, /\* count \*/

MPI\_DOUBLE, /\* datatype \*/

MPI\_PROD, /\* operator \*/

0, /\* root rank \*/

MPI\_COMM\_WORLD); /\* communicator \*/

/\* give the output to the user \*/

**if**(myRank==0)**{**

printf("The factorial of %d is %lf, and was calculated using %d processes\n",fact,total,size);

**}**

/\* shut down MPI \*/

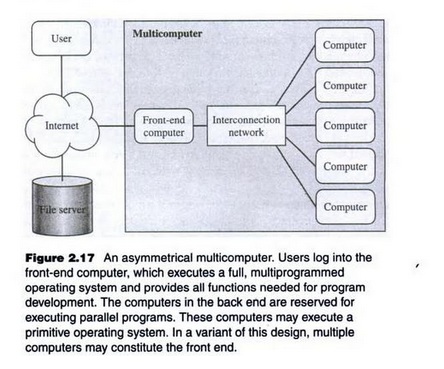
MPI\_Finalize();

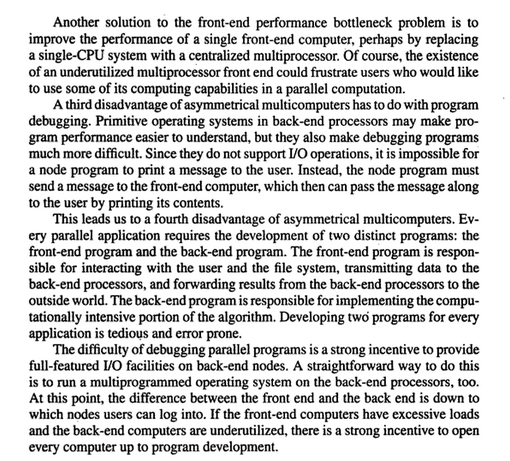
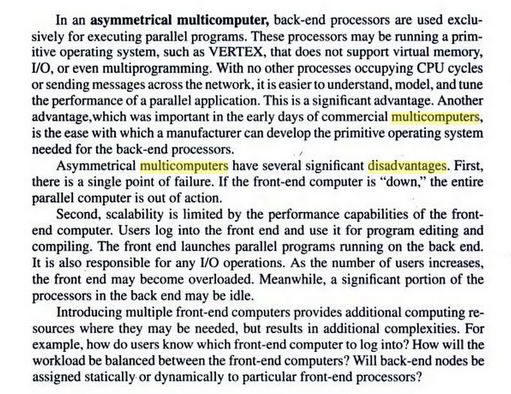
**return** 0;

**}**

<http://mrbook.org/school/parallel/factorial_c.html>







<https://books.google.co.in/books?id=ukb6K3MjhgEC&pg=PA50&lpg=PA50&dq=asymmetric+multicomputers+advantages+disadvantages&source=bl&ots=yyFAk90o8n&sig=wSv6OfnEiEQuM2PqLlHU_hIDPgE&hl=en&sa=X&ved=0ahUKEwjcvJL9qbjXAhWDro8KHVtjDncQ6AEIXzAL#v=onepage&q=asymmetric%20multicomputers%20advantages%20disadvantages&f=false>

