PARALLEL PROGRAMMING RECIPE

The process of development of parallel programs can be summarized as follows:

1. Pick up a particular problem of interest;

2. Conceptualize the solution;

3. Split this solution into components to be executed simultaneously as cooperating processes;

4. Code each component;

5. Arrange components in groups;

6. Allocate to each group a separate processor of suitable type;

7. Execute simultaneously all components, noting overall run time.

A question might be asked:

WHY GROUPS?

Another nasty question:

We hope for speedup > 1.
What if the application of this recipe yields speedup < 1 ?
In particular, speedup = 0 implies deadlock!

WHY PROCESS GROUPS?

We may face two possible situations:

1. If the number of processes <= number of processors and
   it is possible to satisfy all processes' needs for processor types,

   then
   Allocate each process to a separate processor,
   Keeping all processor types compatible with processes' needs

2. else
   We must have groups of processes,
   Each group being allocated a single processor, and
   All processes within each group running in a time-sharing mode.

NOTE 1: We focus on situation (2), considering situation (1) as special case of situation (2).

NOTE 2: In general, no brute-force processor allocation approach is feasible, because there are N^P ways to allocate P processes among N processors.

However, this approach is feasible for a small N and P, and the evolution seems to "know" it…

THE PROCESSOR ALLOCATION PROBLEM

Given:

1. A computer with N processors (of possibly differing characteristics like speed, amount of local memory, presence or absence of floating point facilities, graphics, FFT, etc.)

and

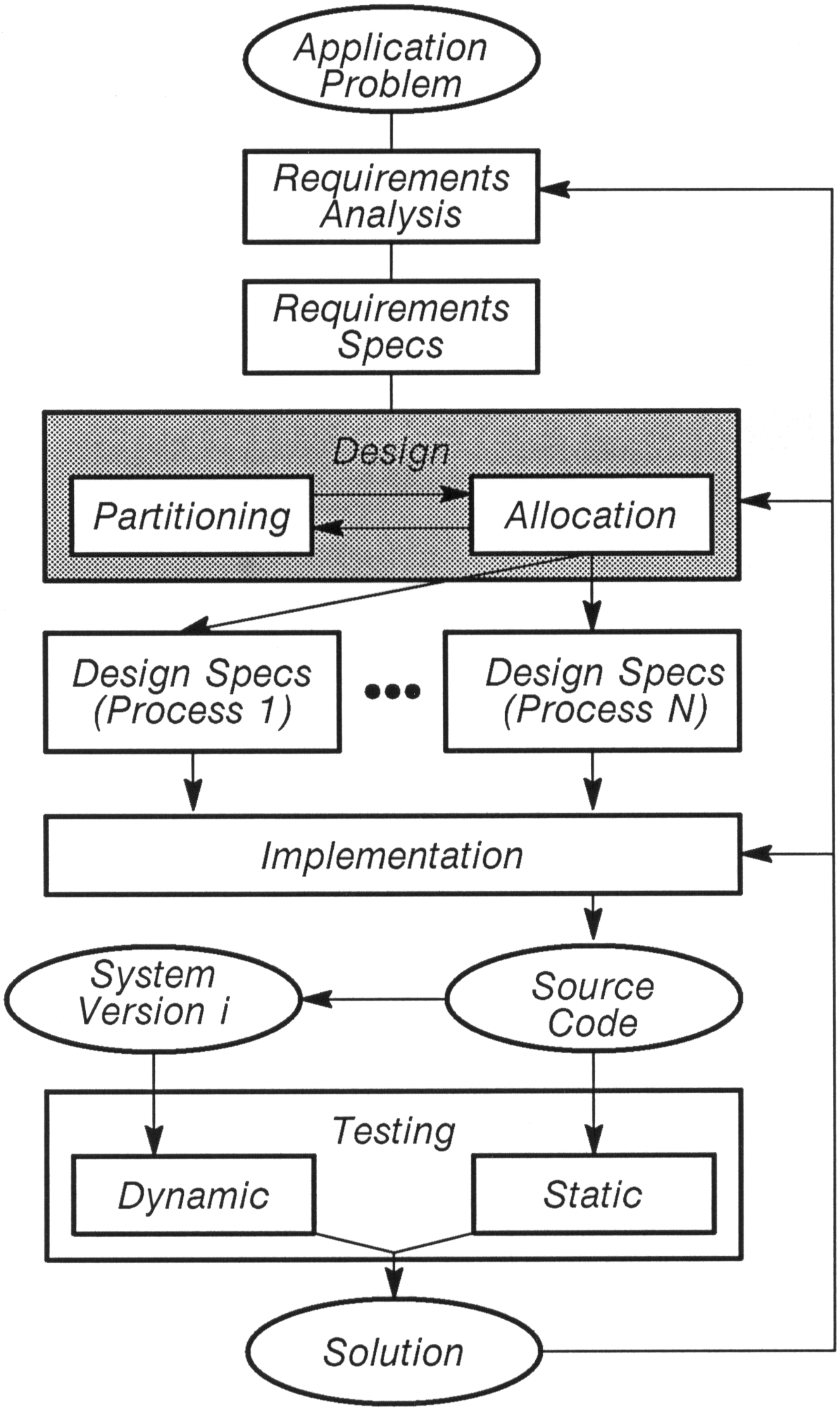
2. An algorithm consisting of a mix of P processes (of possibly differing processor type preferences)

Decide:

1. Which process(es) to run on which processor(s), when, and for how long?
2. On the order sequence of such assignments;
3. On the computer architecture, facilitating interprocess communication.

Our measure of success will be calculated according to some selected criterion (like speedup, node efficiency, area efficiency, etc.)

DISTRIBUTED SOFTWARE LIFE CYCLE



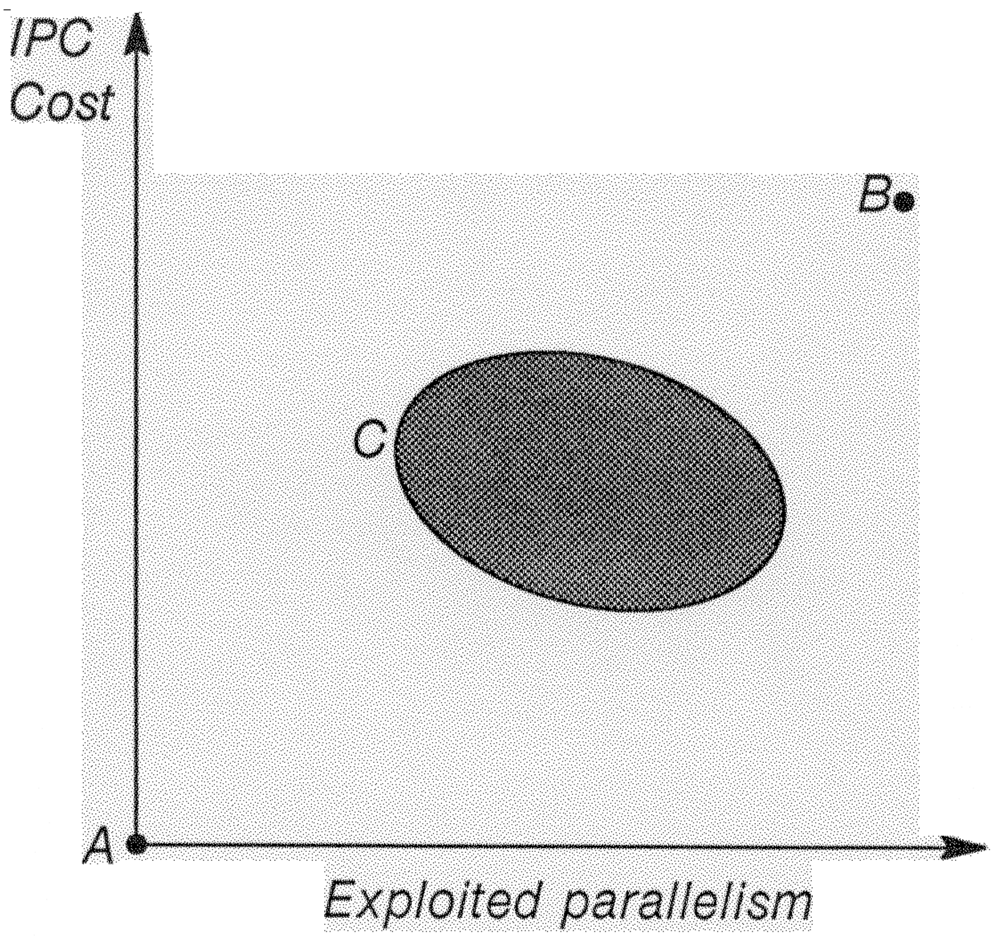
PARTITIONING CONSIDERATIONS

OBJECTIVES:

- Minimize interprocess communication
- Exploit potential concurrency
- Limit sizes of processes

DIFFICULTIES:

- How to measure effectiveness before allocation?
- Partitioning criteria conflicts.



TASK ALLOCATION CONSIDERATIONS

Combinatorially, if we have P processes and N processors, there are N^P possible allocations (or more, if we allow replication to enhance fault tolerance)

Allocation effectiveness depends on the allocation GOAL, like:

- Minimize total IPC cost
- Minimize total computation and IPC cost
- Minimize completion time
- Minimize load imbalance
- Maximize system reliability

Feasible allocations must meet system constraints, like:

- Memory capacity
- Processing time limits

