4-th Workshop on Practical Use of Colored Petri Nets and the CPN Tools,
Aarhus, Denmark, August 28–30, 2002

PERFORMANCE STUDY OF DISTRIBUTED GENERATION OF STATE SPACES USING COLORED PETRI NETS

W.M. Zuberek

Department of Computer Science
Memorial University
St.John's, Canada A1B 3X5

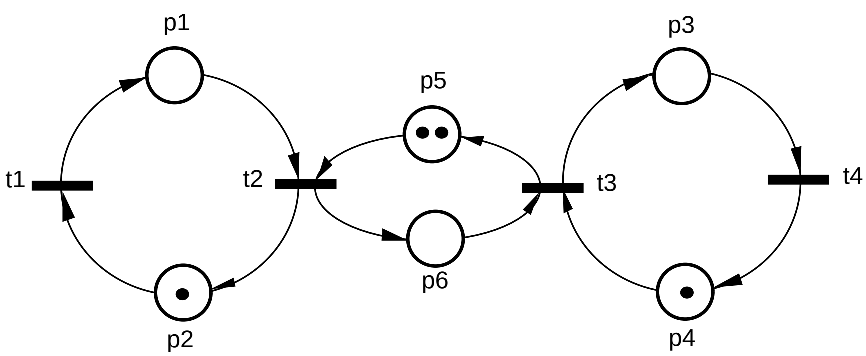
1. TIMED PETRI NETS
2. SEQUENTIAL GENERATION ALGORITHM
3. DISTRIBUTED GENERATION ALGORITHM
4. PETRI NET MODEL
5. PERFORMANCE RESULTS
6. CONCLUDING REMARKS

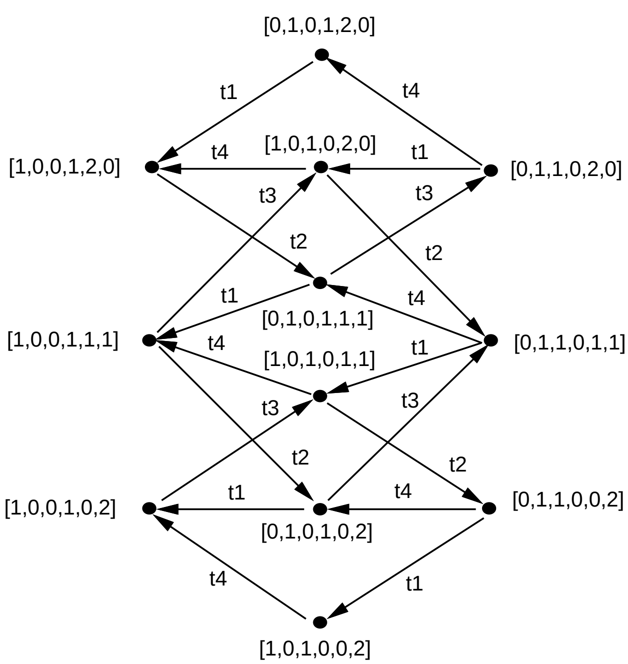
A timed Petri net is $\mathcal{T} = (\mathcal{M}, c, f)$ where:

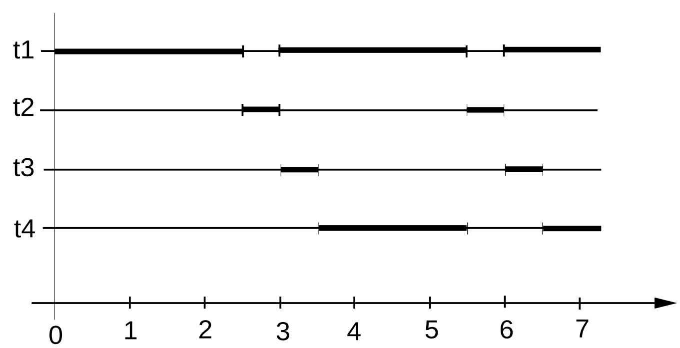
$\mathcal{M}$ is a marked Petri net, $\mathcal{M} = (P, T, A, m_0)$,

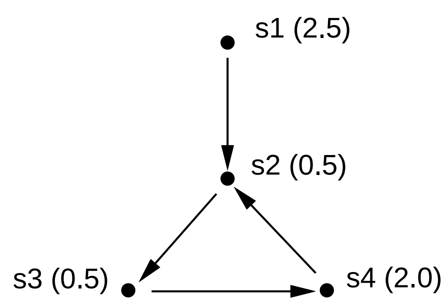
c is a conflict–resolution function, $c : T \to [0, 1]$,

f is a firing–time function, $f : T \to \mathbf{R}^+$.

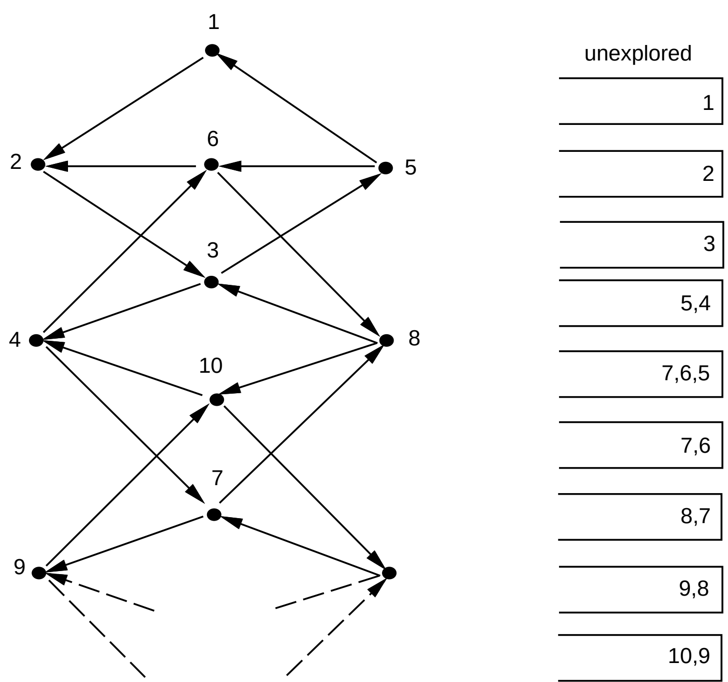




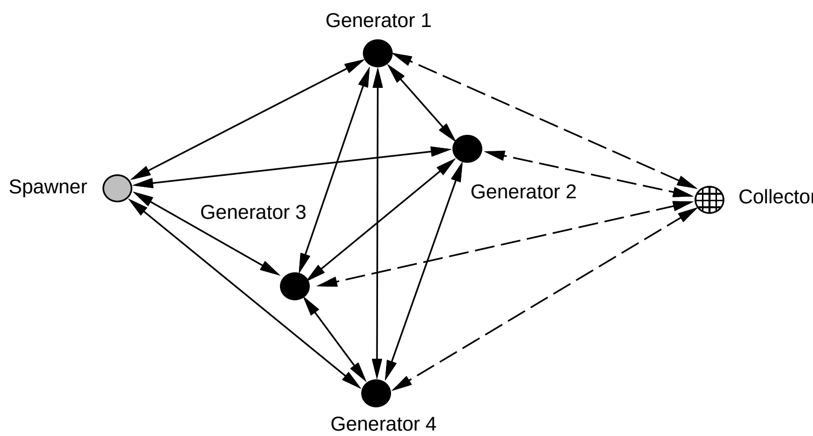




```
State space generation:
var States := ∅;              (* set of states *)
    unexplored := ∅;          (* queue of states *)
begin
    for each s in IntitalStates(m_0) do
        States := States ∪ {s};
        insert(unexplored, s)
    endfor;
    while nonempty(unexplored) do
        state := remove(unexplored);
        for each s in NextStates(state) do
            if s ∉ States then
                States := States ∪ {s};
                insert(unexplored, s)
            endif
        endfor
    endwhile
end.
```

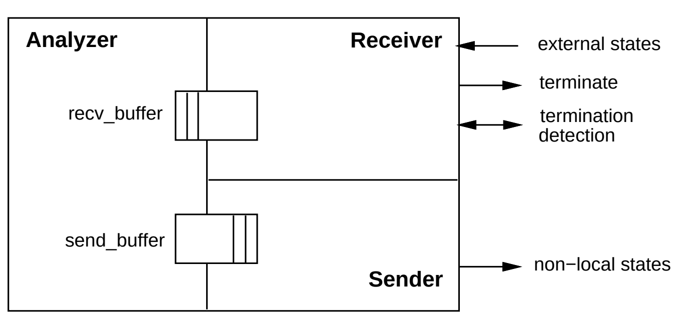


Sequential generation.



Distributed generation.

```
Spawner:
var m_0;                      (* initial marking *)
    k;                        (* the number of hosts *)
    processor_table[];        (* processor identifiers *)
begin
    input virtual machine description and model specification;
    spawn Collector on this_host;
    for i := 1 to k do
        processor_table[i] := spawn Generator on host[i]
    endfor;
    broadcast(process_table);
    for each state in InitialStates(m_0) do
        send(processor_table[region(state)], state)
    endfor
end.
```

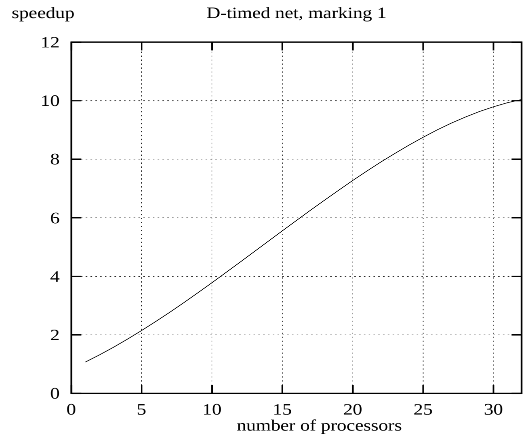


The structure of a *Generator*.

```
Analyzer_i:
var States_i := ∅;            (* set of states *)
    unexplored := ∅;          (* queue of states *)
    cont := true;             (* continuation flag *)
begin
    spawn Receiver, Sender on this_host;
    while cont do
        if empty(recv_buffer) ∧ nonempty(unexplored) then
            state := remove(unexplored);
            new := true
        else
            state :=get(recv_buffer);
            if state = null then
                cont :=false
            else
                new := state ∉ States_i;
                if new then
                    States_i := States_i ∪ {state}
                endif
            endif
        endif;
        if cont ∧ new then
            for each s in NextStates(state) do
                if region(s) = i then
                    if not s ∉ States_i then
                        States_i := States_i ∪ {s};
                        insert(unexplored, s)
                    endif
                else
                    put(send_buffer, s)
                endif
            endfor
        endif
    endwhile
end.
```

$N_{nodes} = 14,487$

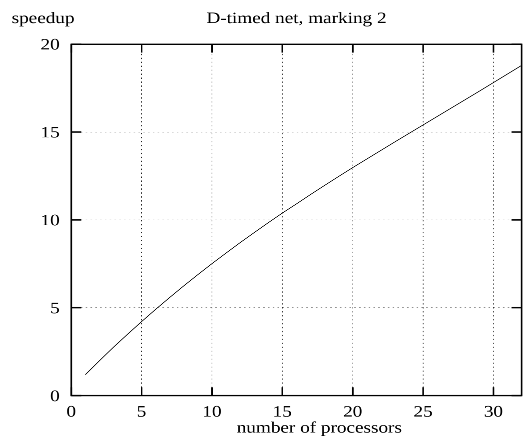
$N_{arcs} = 26,675$



Speedup for Example–1a.

$N_{nodes} = 46,729$

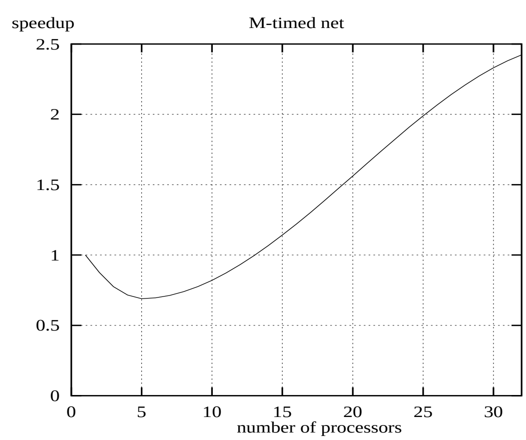
$N_{arcs} = 92,253$



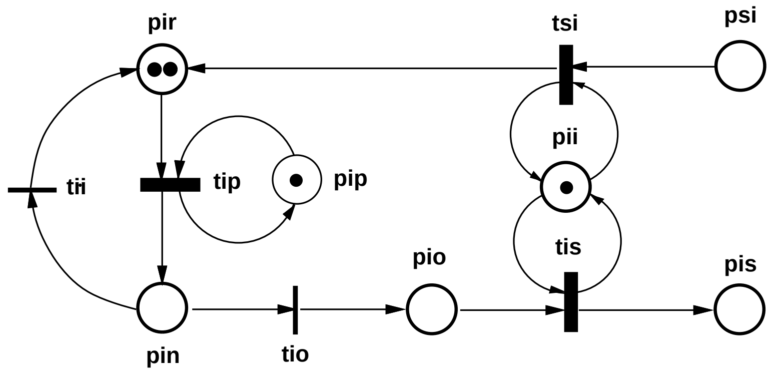
Speedup for Example–1b.

$N_{nodes} = 27,399$

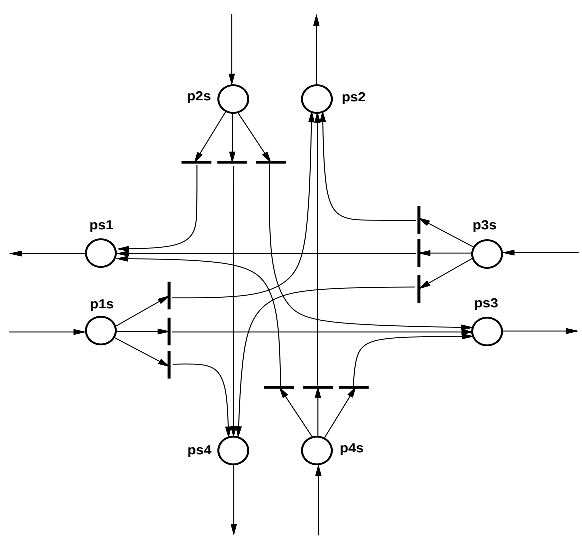
$N_{arcs} = 197,337$



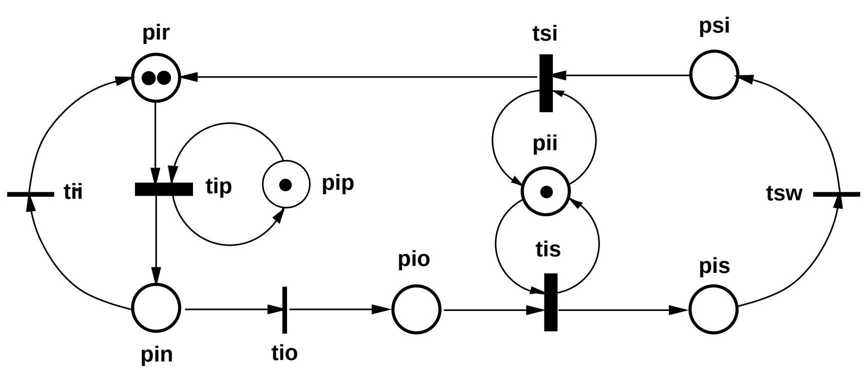
Speedup for Example–2.



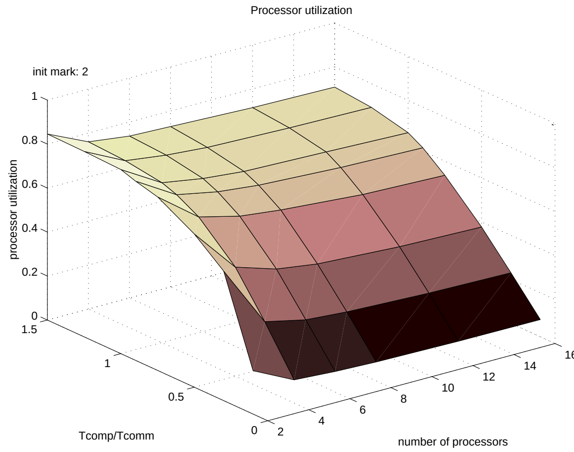
Petri net model of a processor and its link.



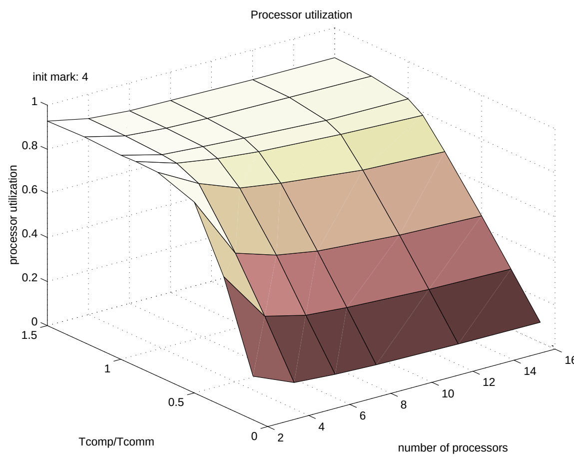
Petri net model of a switch.



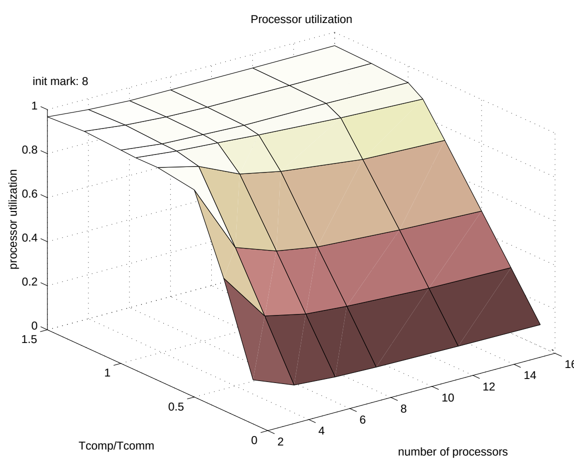
Colored Petri net model of a cluster.



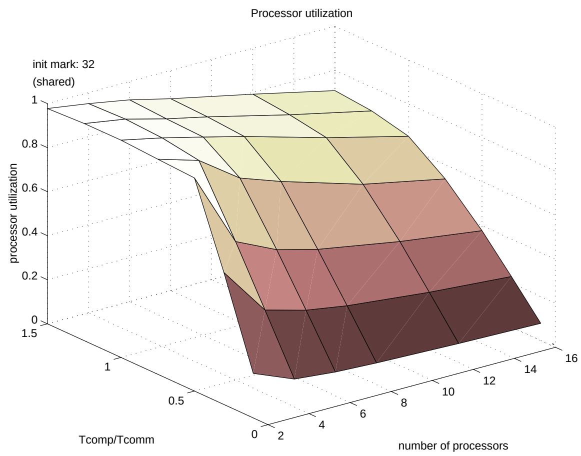
Utilization of processors, low proportional load.



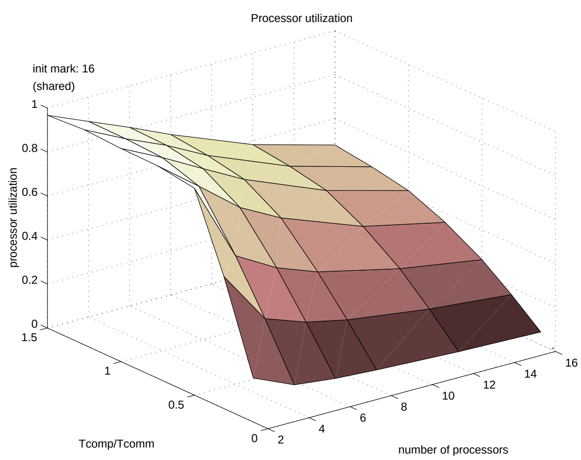
Utilization of processors, medium proportional load.



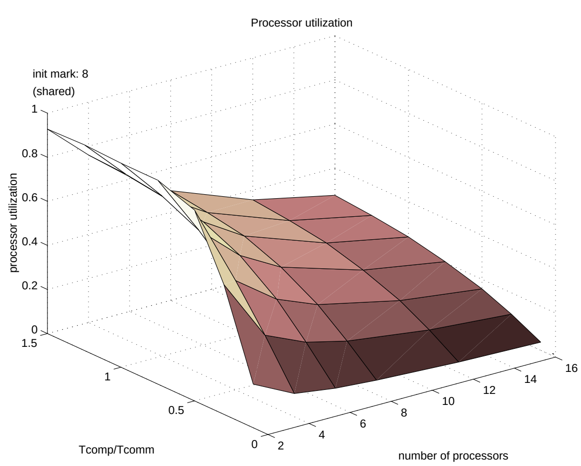
Utilization of processors, high proportional load.



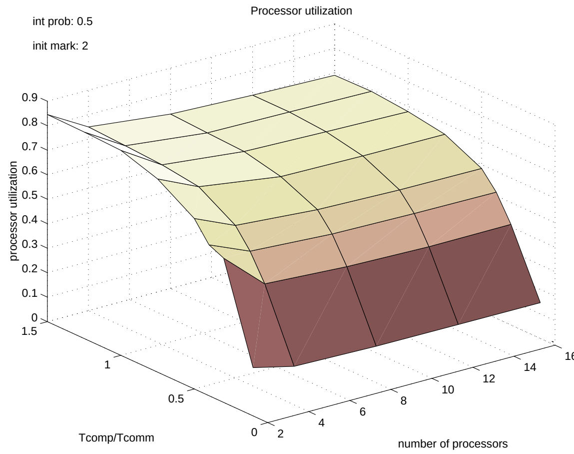
Utilization of processors, shared high load.



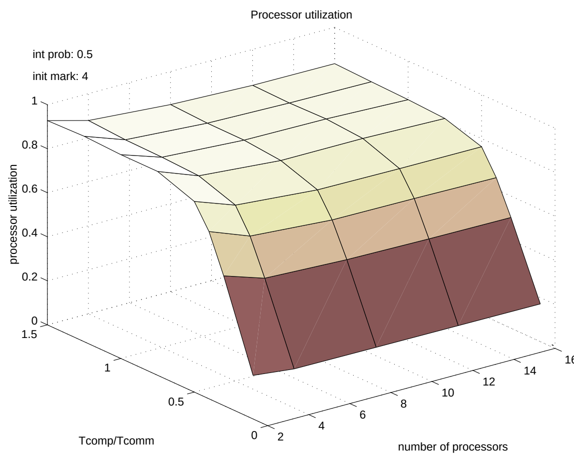
Utilization of processors, shared medium load.



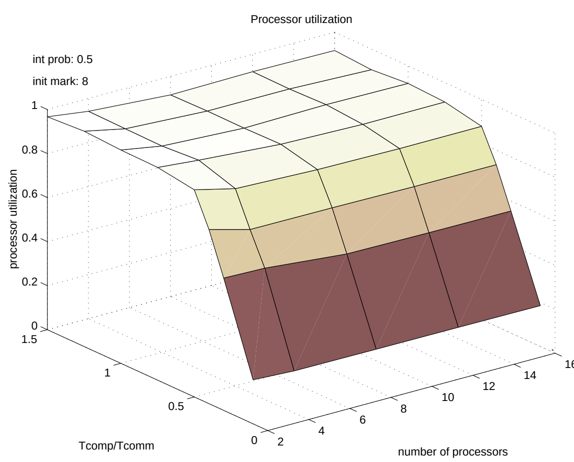
Utilization of processors, shared low load.



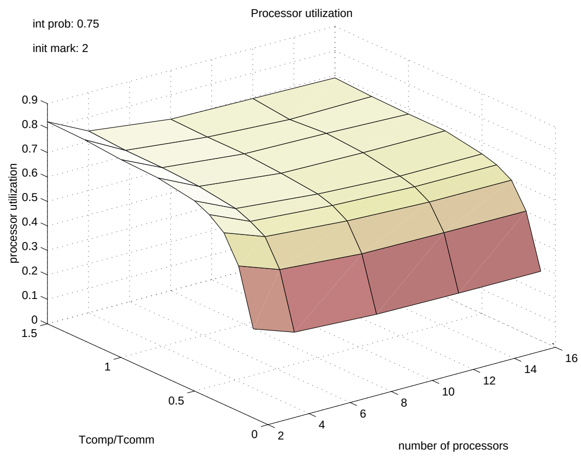
Utilization of processors, low load, $p_{local} = 0.5$.



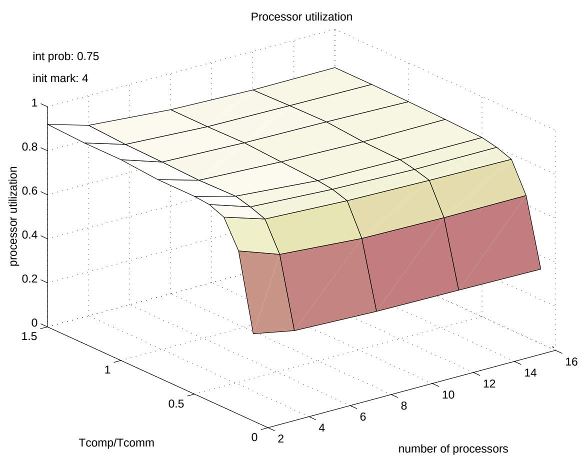
Utilization of processors, medium load, $p_{local} = 0.5$.



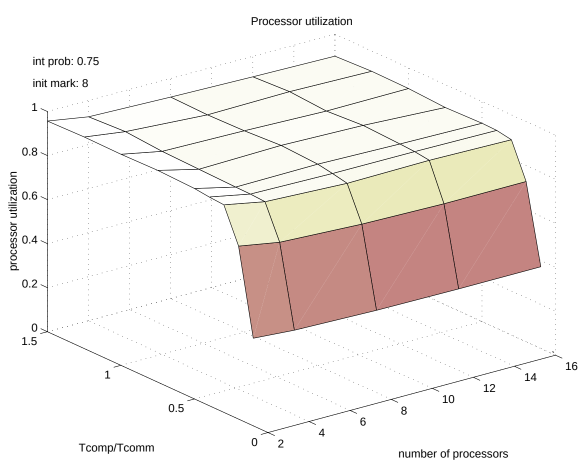
Utilization of processors, high load, $p_{local} = 0.5$.



Utilization of processors, low load, $p_{local} = 0.75$.



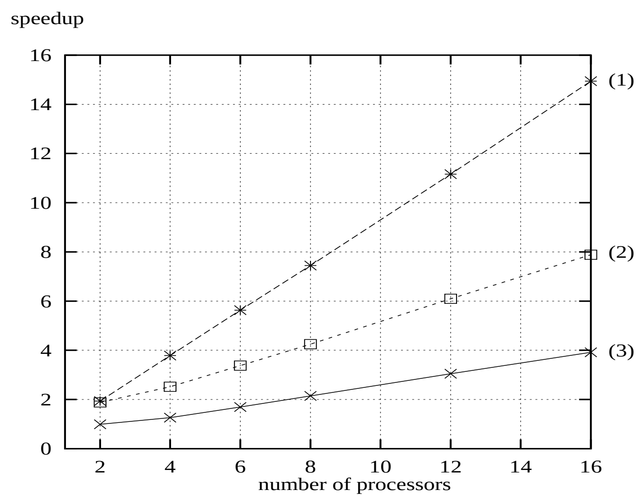
Utilization of processors, medium load, $p_{local} = 0.75$.



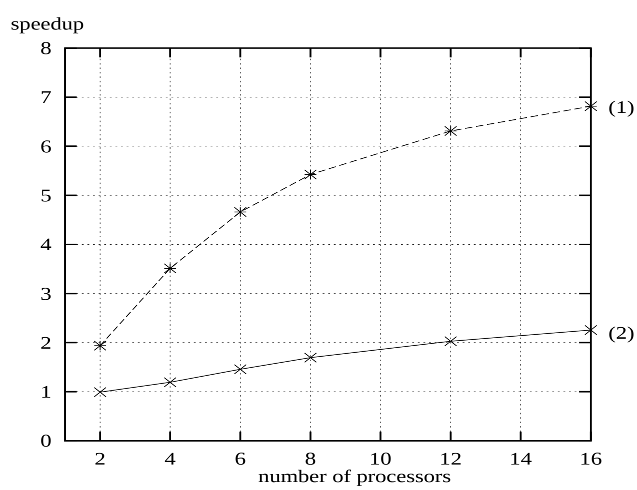
Utilization of processors, high load, $p_{local} = 0.75$.

$$S(k_p) = \frac{T(1)}{T(k_p)}; \qquad T(k_p) = \frac{T(1)}{k_p}\frac{1}{u_p(k_p)}$$

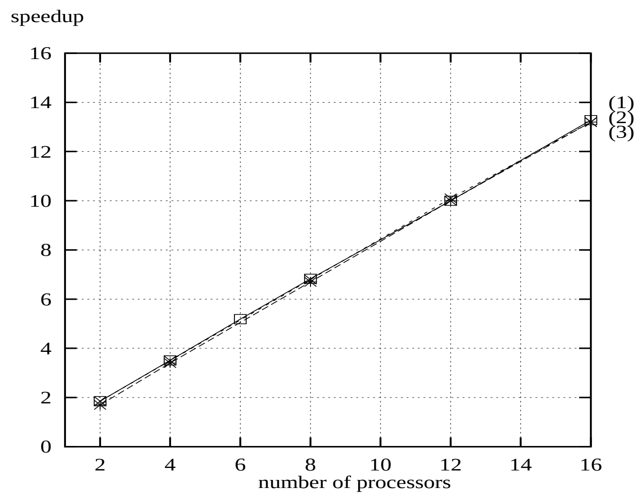
$$S(k_p) = k_p u_p(k_p)$$



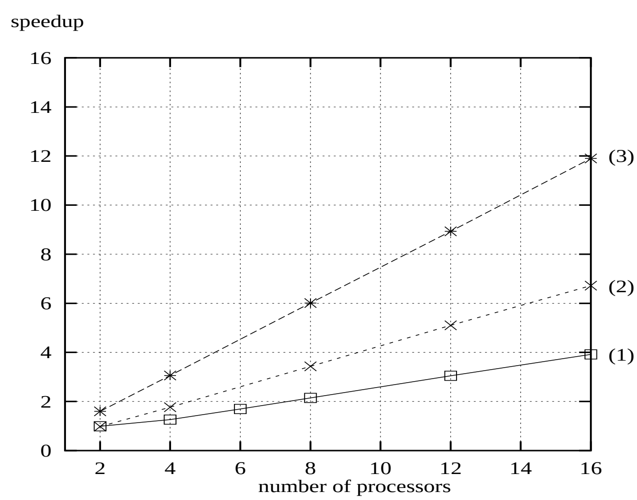
Speedup as a function of k_p for proportional load;
(1) $r_{comp/comm} = 1$, (2) $r_{comp/comm} = 0.5$, (3) $r_{comp/comm} = 0.25$.



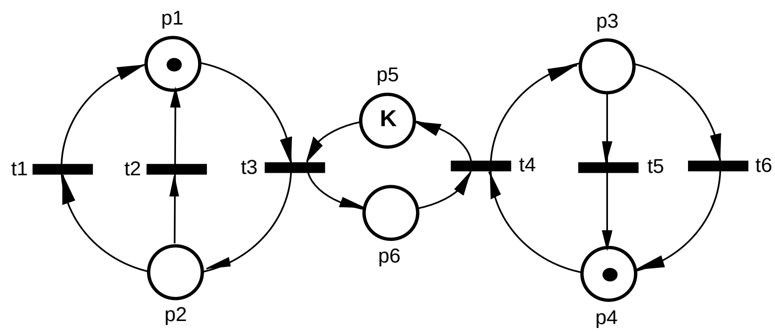
Speedup as a function of n_p for shared load;
(1) $r_{comp/comm} = 1$, (2) $r_{comp/comm} = 0.25$.



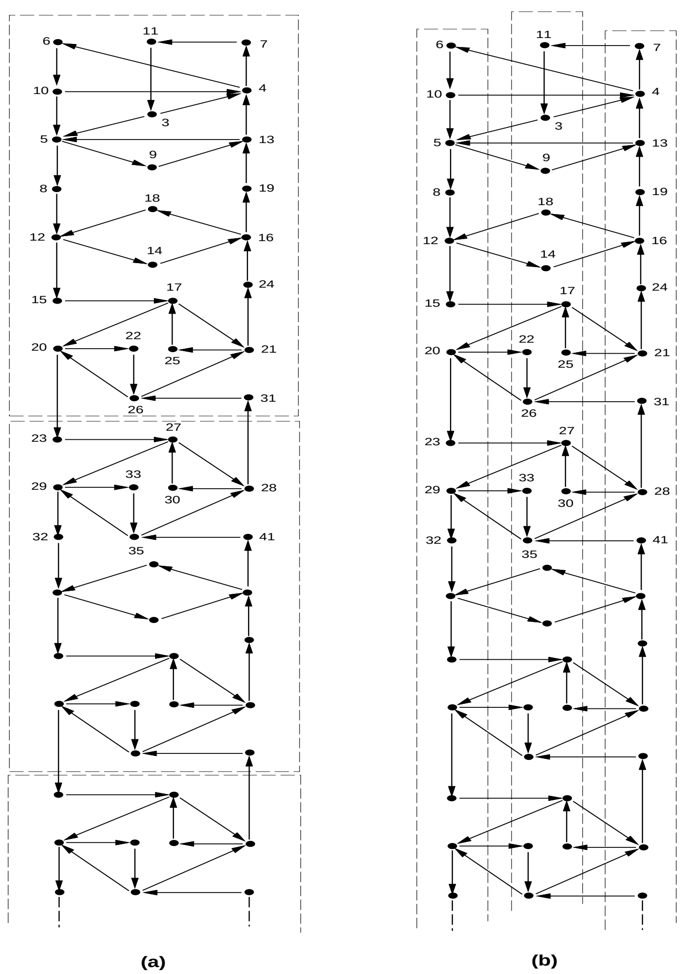
Speedup as a function of k_p for $r_{comp/comm} = 1.0$;
(1) $p_{local} = 1/k_p$; (2) $p_{local} = 0.5$; (3) $p_{local} = 0.75$.



Speedup as a function of k_p for $r_{comp/comm} = 0.25$;
(1) $p_{local} = 1/k_p$; (2) $p_{local} = 0.5$; (3) $p_{local} = 0.75$.



Net model of a bounded–buffer synchronization.



Part of the reachability graph with two different partitioning functions.

CONCLUDING REMARKS:

- THE RATIO OF COMPUTATION TO COMMUNICATION, $r_{comp/comm}$, (IF IT CAN BE REASONABLY DETERMINED) IS A MAJOR FACTOR AFFECTING THE PERFORMANCE OF DISTRIBUTED APPLICATIONS
- IF $r_{comp/comm} > 1$, THE APPLICATION'S PERFORMANCE IS PRACTICALLY INSENSITIVE TO COMMUNICATION DELAYS
- IF $r_{comp/comm} < 1$, THE LOCALITY OF THE FUNCTION PARTITIONING THE STATE SPACE DETERMINES THE APPLICATION'S PERFORMANCE
- THE FUNCTION PARTITIONING THE STATE SPACE MAY NEED TO BE TUNED DYNAMICALLY TO IMPROVE ITS LOCALITY AND UNIFORMITY OF STATE DISTRIBUTION AMONG THE PARTITIONS (WHICH WILL INCREASE COMMUNICATION)
- SCALABILITY OF DISTRIBUTED APPLICATIONS IS, IN GENERAL, A NONTRIVIAL ISSUE
- THE INFLUENCE OF MANY OTHER FACTORS NEEDS TO BE TAKEN INTO ACCOUNT (E.G., NON–UNIFORM DISTRIBUTION OF STATES AMONG THE REGIONS, DISTRIBUTIONS OF EXECUTION TIMES, VARIABLE NUMBERS OF NEXT STATES, ETC.)