

```

specification Demo4 [g, h] : noexit

library DecNatRepr
endlib

behaviour
  A [g] |[g]| B [g, h] |[h]| C [h]

where
  process A [g] : noexit := TypeOne [g] endproc

  process B [g, h] : noexit := TypeTwo [g, h] endproc

  process C [h] : noexit := TypeOne [h] endproc

  process TypeOne [g] : noexit :=
    (*| delay 1 |*)
    g ? x : DecDigit;
    TypeOne [g]
  endproc

  process TypeTwo [g, h] : noexit :=
    (
      g ! 1 of DecDigit;
      (*| delay 2 |*)
      h ! 2 of DecDigit;
      exit
    []
      h ! 1 of DecDigit;
      (*| delay 3 |*)
      g ! 2 of DecDigit;
      exit
    )
    >> TypeTwo [g, h]
  endproc
endspec

```

Figure 1: LOTOS Example with Behavioural Annotations

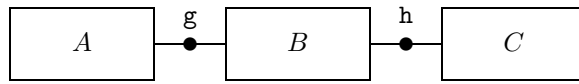


Figure 2:

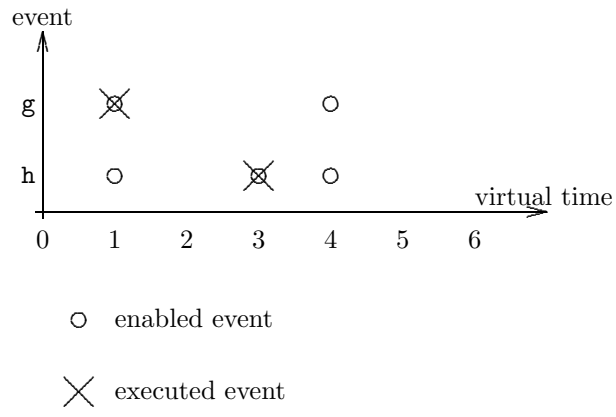


Figure 3:

```

function  $\Omega : (\mathbf{S}^0 \in \mathcal{S}) \mapsto (\mathcal{S} \times \mathcal{E})^+$ 
variables
   $e^i \in \mathcal{E} : i = 0, 1, 2, \dots$ 
   $\mathbf{S}^i \in \mathcal{S} : i = 0, 1, 2, \dots$ 
   $a \in \mathcal{A}; g \in \mathcal{G}; i \in \mathbb{Z}; t \in \mathcal{T}$ 
begin
   $i \leftarrow 0$ 
  loop
     $(t, g, a) \leftarrow \Gamma(\rho(\mathbf{S}^i))$ 
    exit when  $t > T_{\max}$ 
     $e^i \leftarrow (t, g, a)$ 
     $\mathbf{S}^{i+1} \leftarrow \Theta(\mathbf{S}^i, e^i)$ 
     $i \leftarrow i + 1$ 
  end loop
end
result  $\{(\mathbf{S}^0, e^0), (\mathbf{S}^1, e^1), (\mathbf{S}^2, e^2) \dots\}$ 

```

Figure 4: Sequential Execution Algorithm

```

function  $\Omega : (\mathbf{S}^0 \in \mathcal{S}) \mapsto (\mathcal{S} \times \mathcal{E})^+$ 
variables
   $e^i \in \mathcal{E} : i = 0, 1, 2, \dots$ 
   $\mathbf{S}^i \in \mathcal{S} : i = 0, 1, 2, \dots$ 
   $a \in \mathcal{A}; A \in 2^{\mathcal{G}}; g \in \mathcal{G}; i \in \mathbb{Z}; \mathbf{R} \in \mathcal{O}; \widehat{\mathbf{S}} \in \mathcal{S}; t \in \mathcal{T}$ 
begin
   $i \leftarrow 0$ 
  loop
    phase 1:
     $\mathbf{R} \leftarrow \rho(\mathbf{S}^i)$ 
     $A \leftarrow \Gamma^*(\mathbf{R})$ 
     $\widehat{\mathbf{S}} \leftarrow \Theta^*(\mathbf{S}^i, A, \mathbf{R})$ 
    phase 2:
    loop
       $(t, g, a) \leftarrow \Gamma(\rho(\mathbf{S}^i))$ 
      exit when  $t > T_{\max}$  or  $g \notin A$  or  $\mathbf{R}_{(\#g)} \neq (t, a)$ 
       $e^i \leftarrow (t, g, a)$ 
      for each  $p \in \mathcal{P}$ 
        if  $p \in \mathcal{C}_{\#g}$  then
           $\mathbf{S}_{(\#p)}^{i+1} \leftarrow \widehat{\mathbf{S}}_{(\#p)}$ 
        else
           $\mathbf{S}_{(\#p)}^{i+1} \leftarrow \mathbf{S}_{(\#p)}^i$ 
        end if
       $i \leftarrow i + 1$ 
    end loop
    exit when  $t > T_{\max}$ 
  end loop
end
result  $\{(\mathbf{S}^0, e^0), (\mathbf{S}^1, e^1), (\mathbf{S}^2, e^2) \dots\}$ 

```

Figure 5: Parallel Execution Algorithm

```

function  $\Theta^* : (\mathbf{S} \in \mathcal{S}, A \in 2^{\mathcal{G}}, \mathbf{R} \in \mathcal{O}) \mapsto \mathcal{S}$ 
variables
   $\widehat{\mathbf{S}} \in \mathcal{S}$ 
begin
   $\widehat{\mathbf{S}} \leftarrow \mathbf{S}$ 
  for each  $g \in A$ 
    for each  $p \in \mathcal{C}_{\#g}$ 
      variables
         $a \in \mathcal{A}_{\#g}; s, s' \in \mathcal{S}_{\#p}; t \in \mathcal{T}$ 
      begin
         $(t, a) \leftarrow \mathbf{R}_{(\#g)}$ 
         $(\cdot, s, \cdot) \leftarrow \mathbf{S}_{(\#p)}$ 
         $s' \leftarrow \alpha_{\#p}(s, (t, g, a))$ 
         $\widehat{\mathbf{S}}_{(\#p)} \leftarrow (\beta(s', t), s', t)$ 
      end
    end
  end
result  $\widehat{\mathbf{S}}$ 

```

Figure 6: Optimistic Execution Phase

```

function  $\Gamma^* : (\mathbf{R} \in \mathcal{O}) \mapsto 2^{\mathcal{G}}$ 
variables
   $a \in \mathcal{A}; A \in 2^{\mathcal{G}}; g \in \mathcal{G}; i \in \mathbb{Z}; \mathbf{R}' \in \mathcal{O}; t \in \mathcal{T}$ 
begin
   $\mathbf{R}' \leftarrow \mathbf{R}$ 
   $A \leftarrow \emptyset$ 
   $i \leftarrow 0$ 
  loop
     $(t, g, a) \leftarrow \Gamma(\mathbf{R}')$ 
    exit when  $t > T_{\max}$  or  $i = I_{\max}$ 
     $A \leftarrow A \cup \{g\}$ 
    for each  $h \in \mathcal{H}_{\#g}$ 
       $\mathbf{R}'_{(\#h)} \leftarrow (\infty, ?)$ 
     $i \leftarrow i + 1$ 
  end loop
end
result  $A$ 

```

Figure 7: Optimistic Scheduling Function

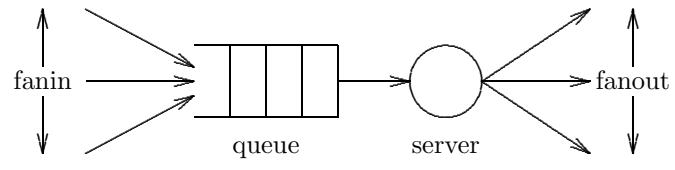


Figure 8: Queue and Server

```
class Node : public LogicalProcess
{
    Time const meanServiceTime;
    unsigned short int const fanin;
    unsigned short int const fanout;
    unsigned short int const load;

    GeometricRV serviceTimeRV;
    NaryRV outputPortRV;

    Deque <Departure> queue;

    void CustomerArrival ();

public:
    void ModelAlpha (OfferVector&) const;
    void ModelBeta (PortNumber, Attribute const&);
};
```

Figure 9: State code

```

void Node::ModelAlpha (PortNumber port, Attribute const&)
{
    if (port < fanin)
    {
        Time const serviceTime = serviceTimeRV .Sample ();
        PortNumber const outputPort = outputPortRV .Sample ();

        Time startTime = time;
        if (!queue .IsEmpty ())
            startTime = queue .TheTail () .TheTime ();

        queue .Enqueue (Departure (startTime + serviceTime, outputPort));
    }
    else
        queue .Dequeue ();
}

```

Figure 10: Behaviour code

```
void Node::ModelBeta (OfferVector& result) const
{
    for (PortNumber i = 0; i < fanin; ++i)
        result [i] = Offer (time, Customer::Unbound ());

    if (!queue .IsEmpty ())
    {
        Departure const& departure = queue .TheHead ();
        result [departure .ThePort ()] =
            Offer (departure .TheTime (), Customer::Bound (1));
    }
}
```

Figure 11: Offers code

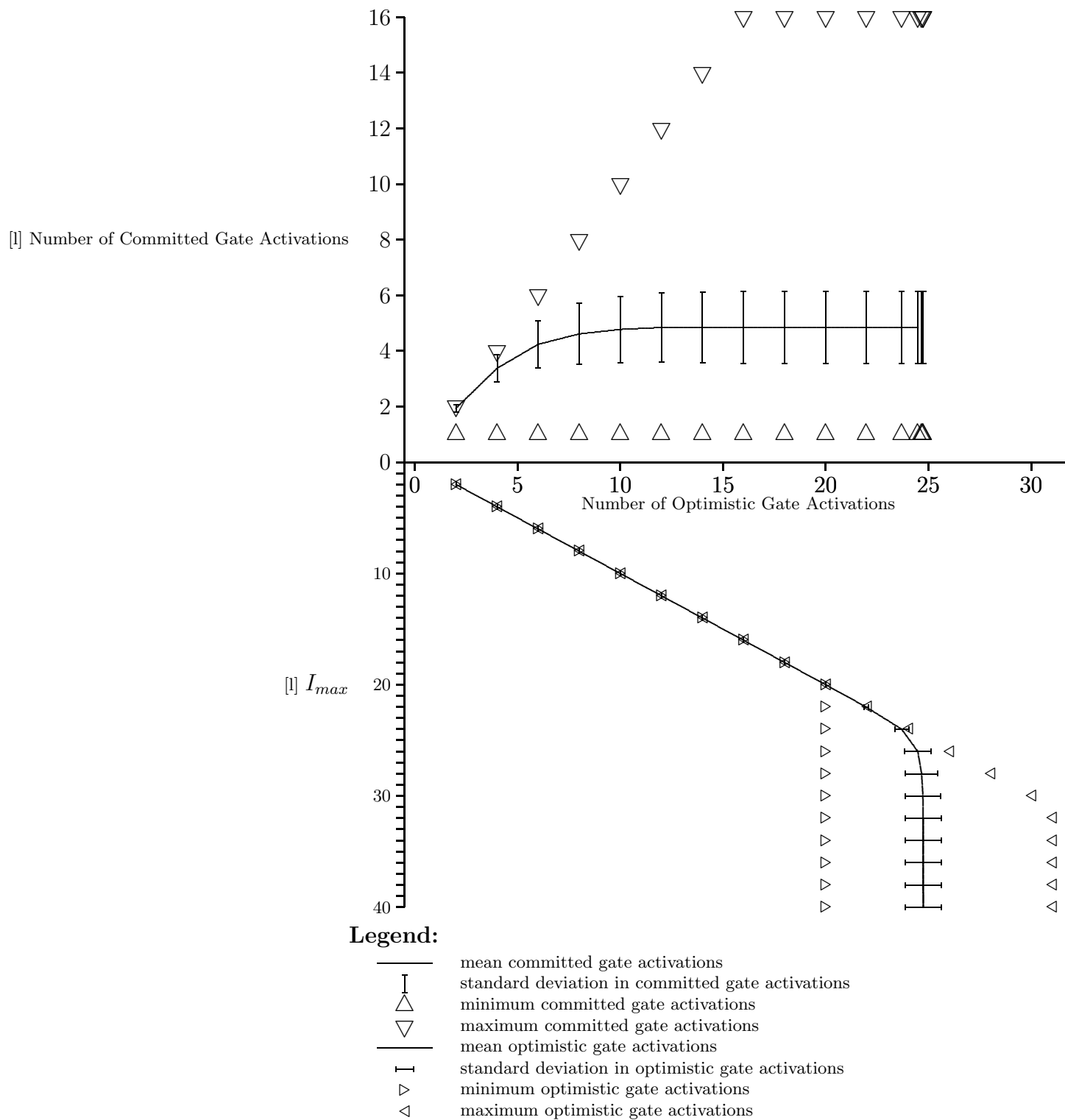


Figure 12: Committed vs. Optimistic Gate Activations, torus.

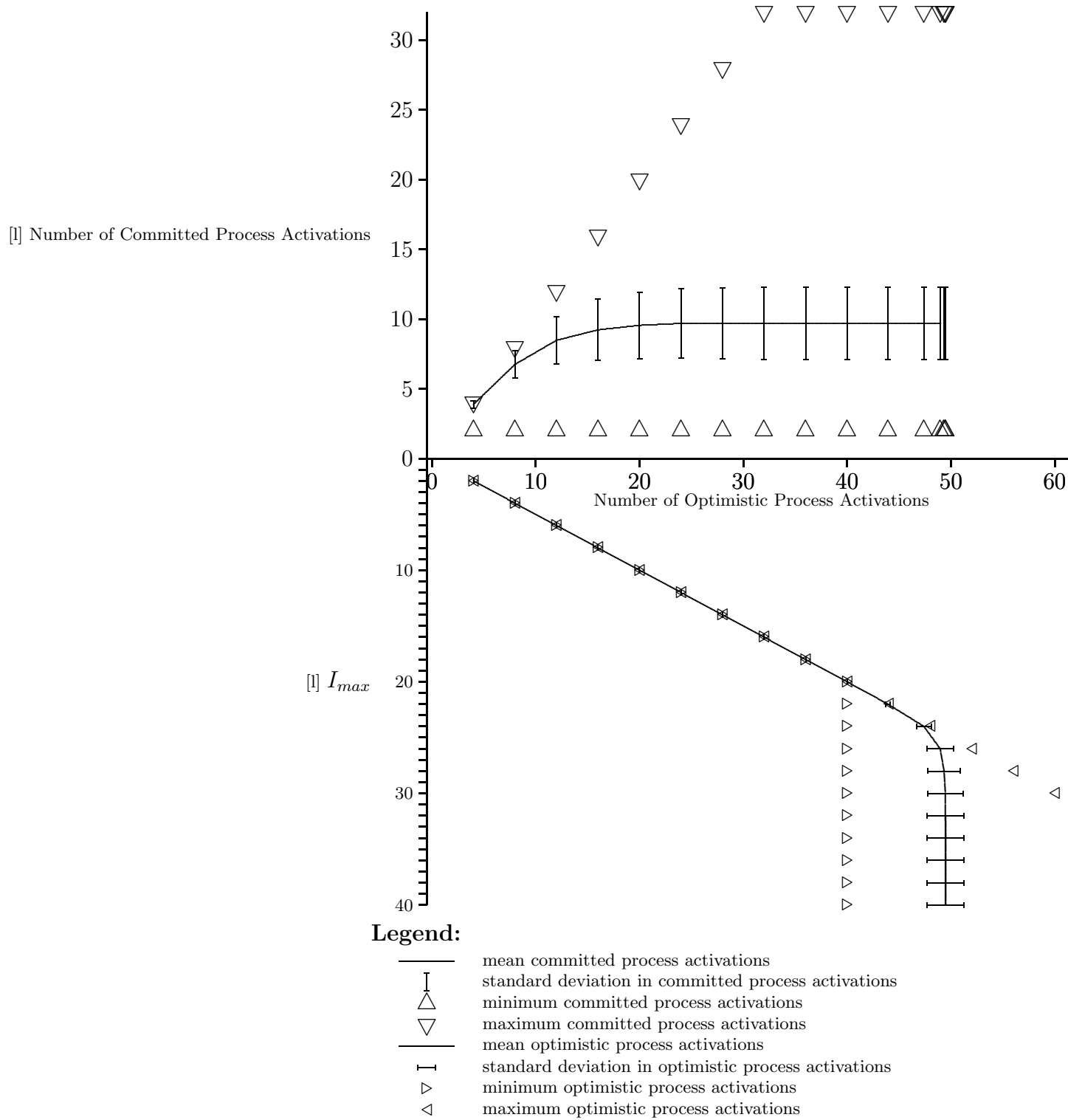


Figure 13: Committed Process vs. Optimistic Process Activations, torus.

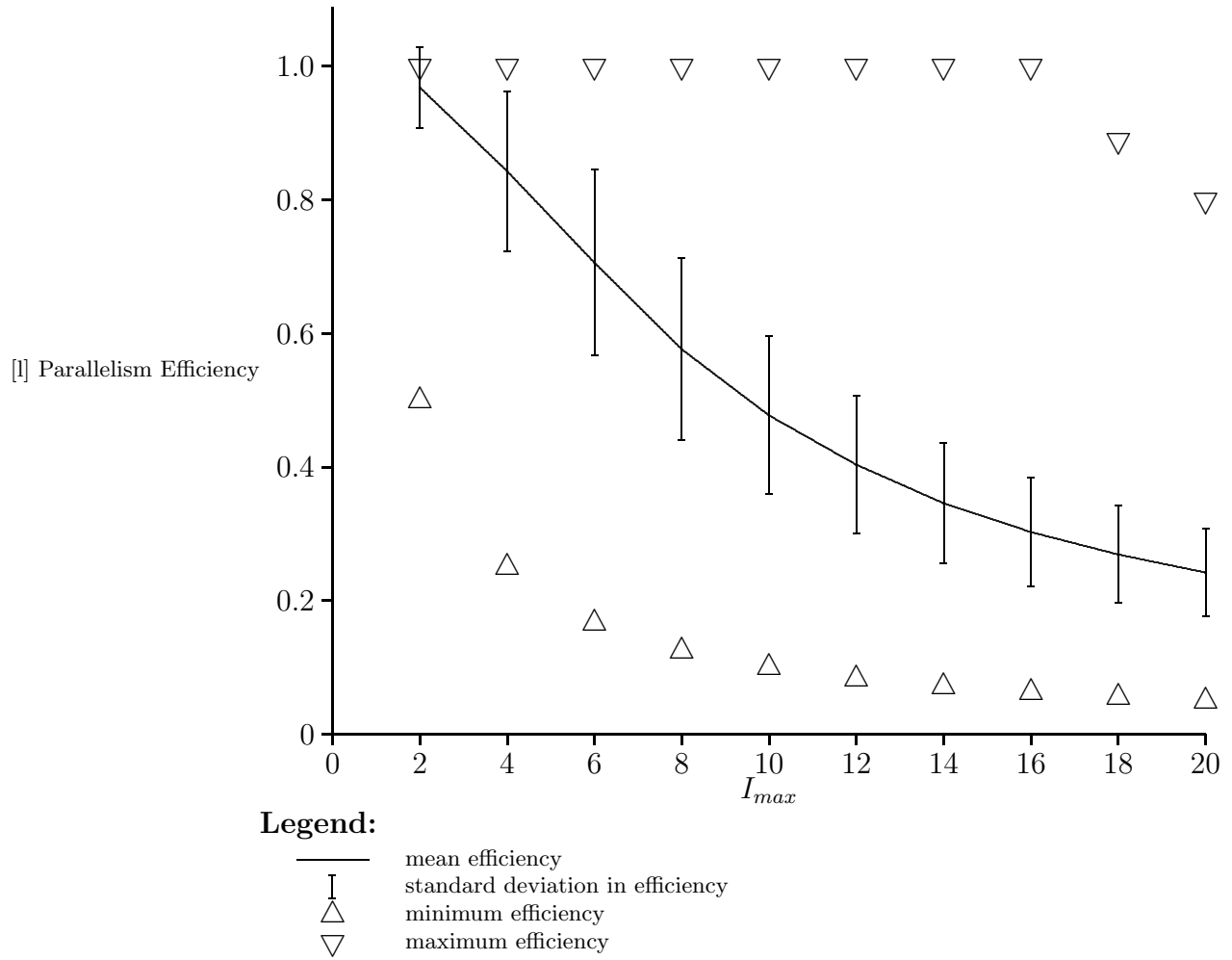


Figure 14: Parallelism Efficiency vs.  $I_{max}$ , torus.

```
class LogicGate : public LogicalProcess
{
    LogicOp const logicOp;
    unsigned short int const fanin;
    Time const delay;

    Array <Level> inputList;
    Level output;

    Deque <Transition> queue;

public:
    void ModelAlpha (OfferVector&) const;
    void ModelBeta (PortNumber, Attribute const&);
};
```

Figure 15: State code

```

void LogicGate::ModelAlpha (PortNumber port, Attribute const& attr)
{
    EdgeAttribute const& inputEdge =
        (EdgeAttribute const&) attr .TheItem ();

    if (port < fanin)
    {
        inputList [port] = inputEdge .TheValue ();
        Level const newOutput = (*logicOp) (inputList);

        if (newOutput != output)
        {
            output = newOutput;
            if (!queue .IsEmpty () &&
                queue .TheTail () .TheTime () == time + delay)
                queue .DequeueTail ();
            else
                queue .Enqueue (Transition (time + delay, Edge (output)));
        }
    }
    else
        queue .Dequeue ();
}

```

Figure 16: Behaviour code

```
void LogicGate::ModelBeta (OfferVector& result) const
{
    for (unsigned short int i = 0; i < fanin; ++i)
        result [i] = Offer (time, EdgeAttribute::Unbound ());

    if (!queue .IsEmpty ())
    {
        Transition const& transition = queue .TheHead ();
        result [fanin] = Offer (transition .TheTime (),
            EdgeAttribute::Bound (transition .TheEdge ()));
    }
}
```

Figure 17: Offers code

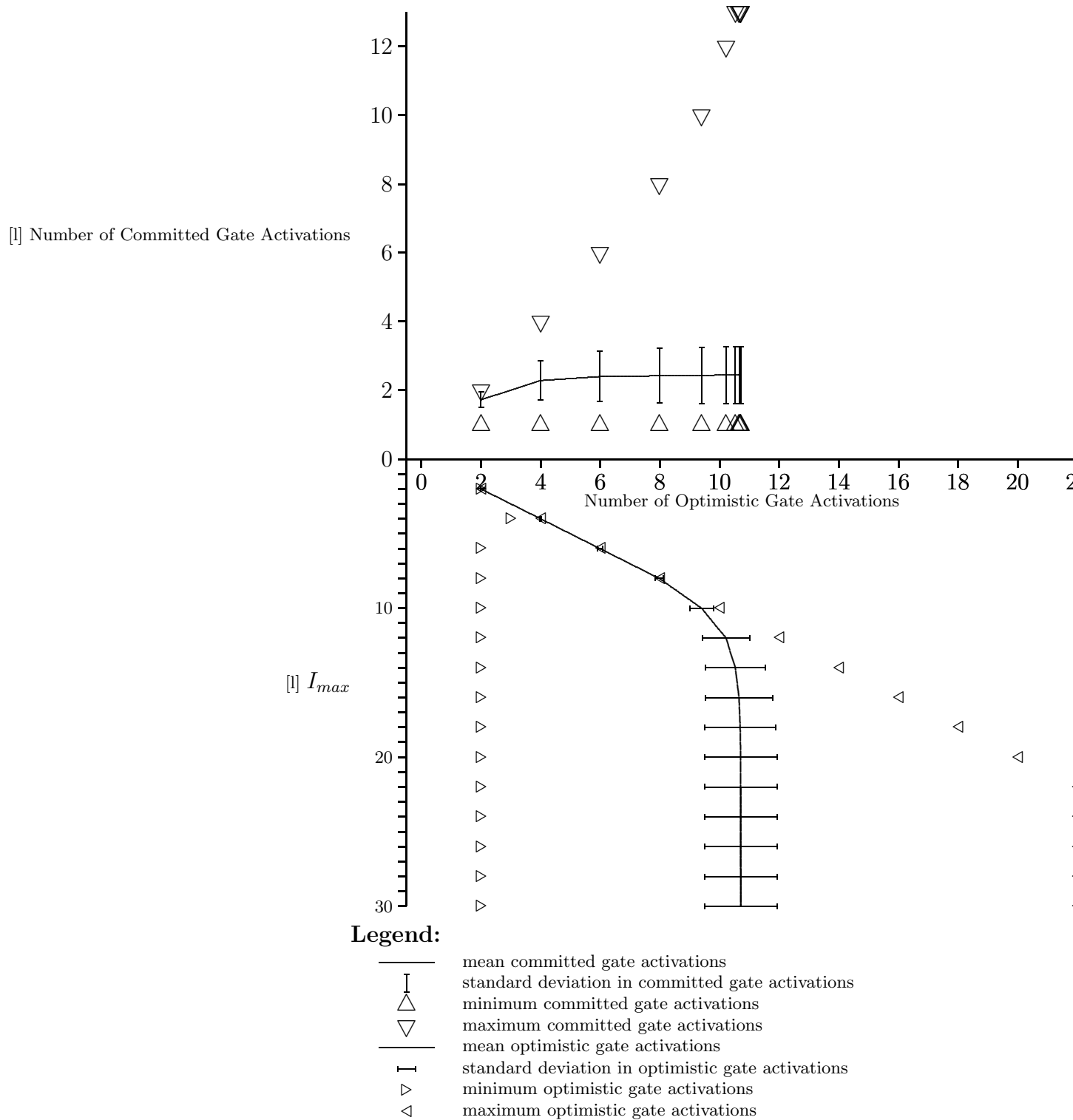


Figure 18: Committed vs. Optimistic Gate Activations, counter.

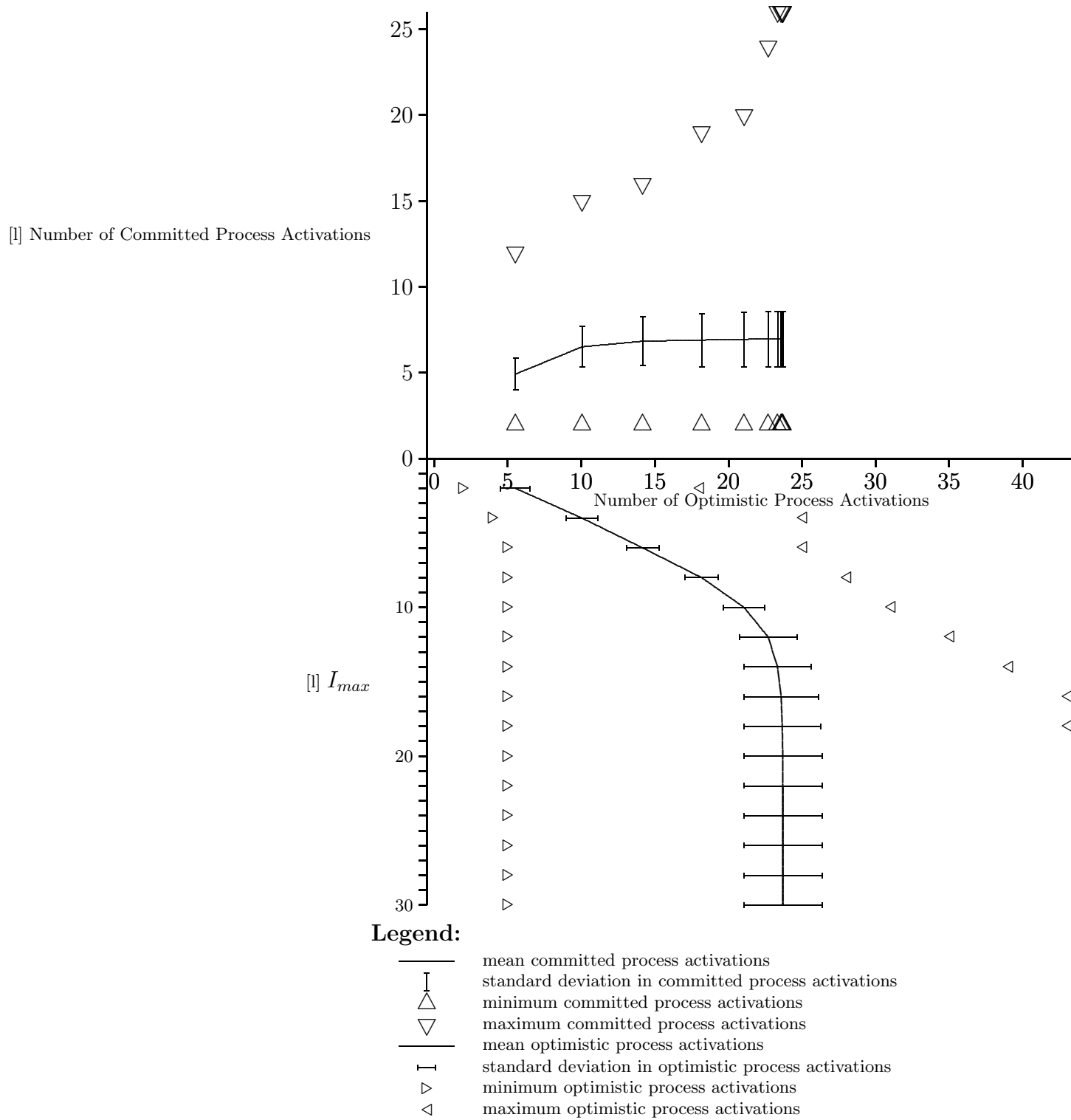


Figure 19: Committed Process vs. Optimistic Process Activations, counter.

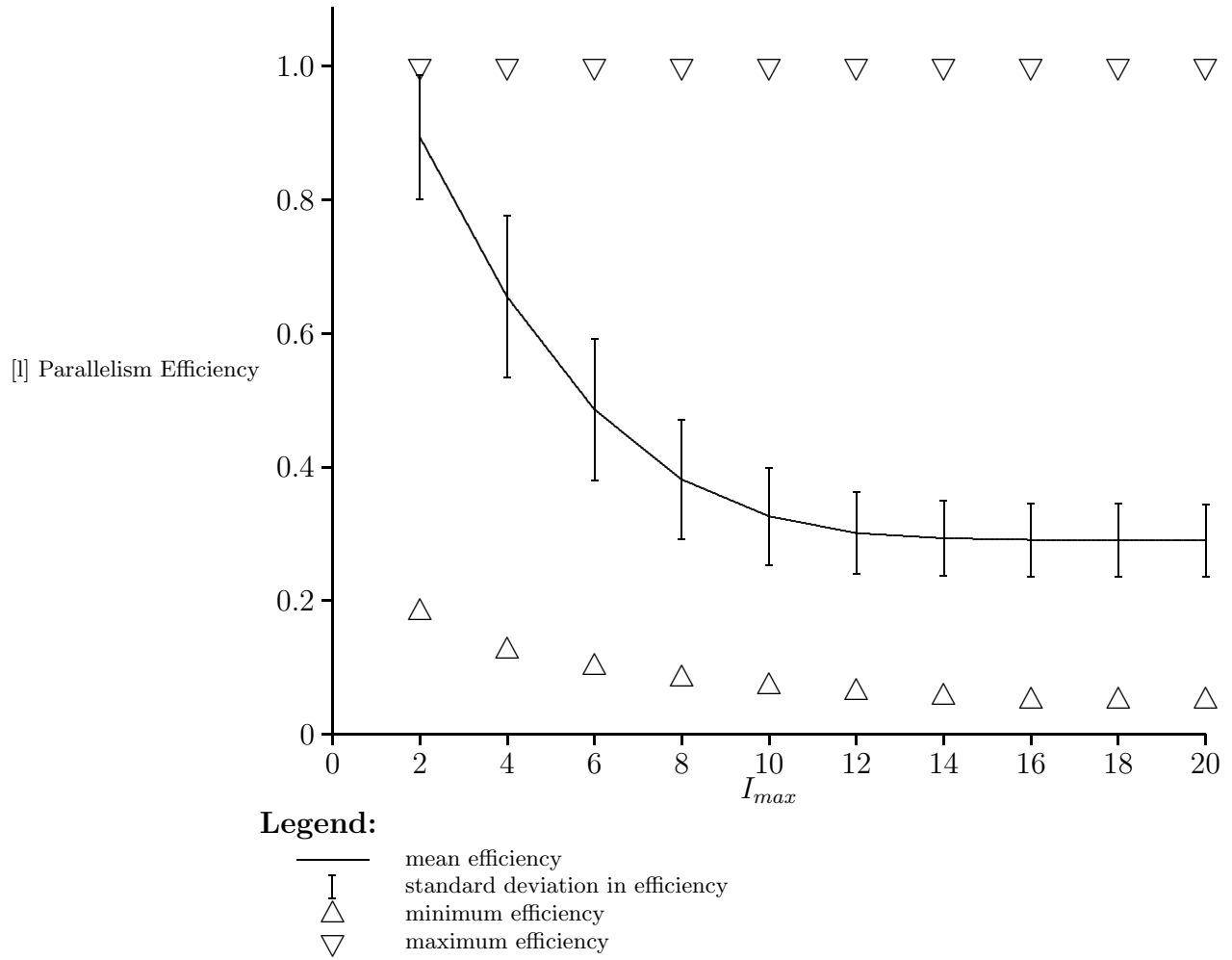


Figure 20: Parallelism Efficiency vs.  $I_{max}$ , counter.