György Hudoba

Exercises on parallel computing

Athens, 13 May - 14 July, 1996.

Some subjects of parallel computing:

computational hunger problems, having inherent parallelism, such as:

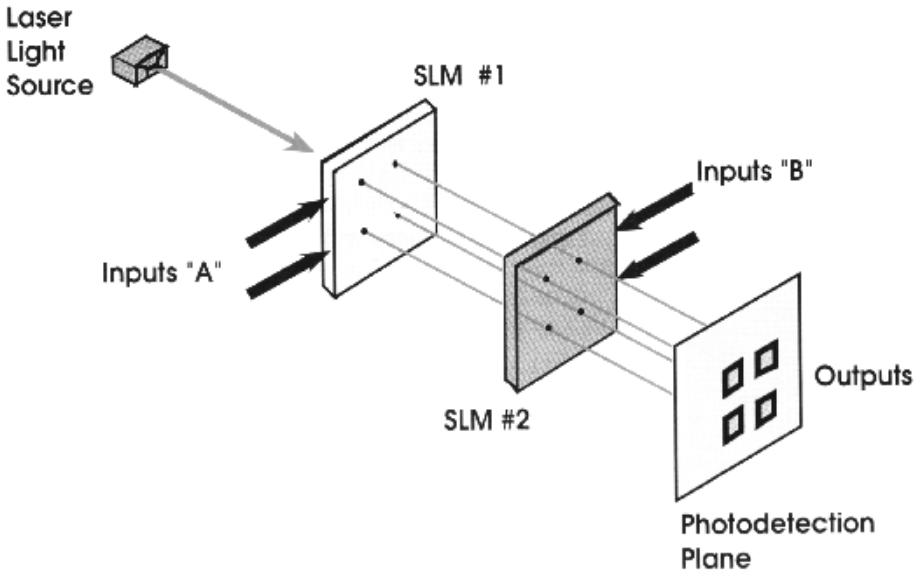
- weather forecast
- modelling,
- image processing,
- modelling different structures, before their have a final hardware realisation.

.

Parallel computers:

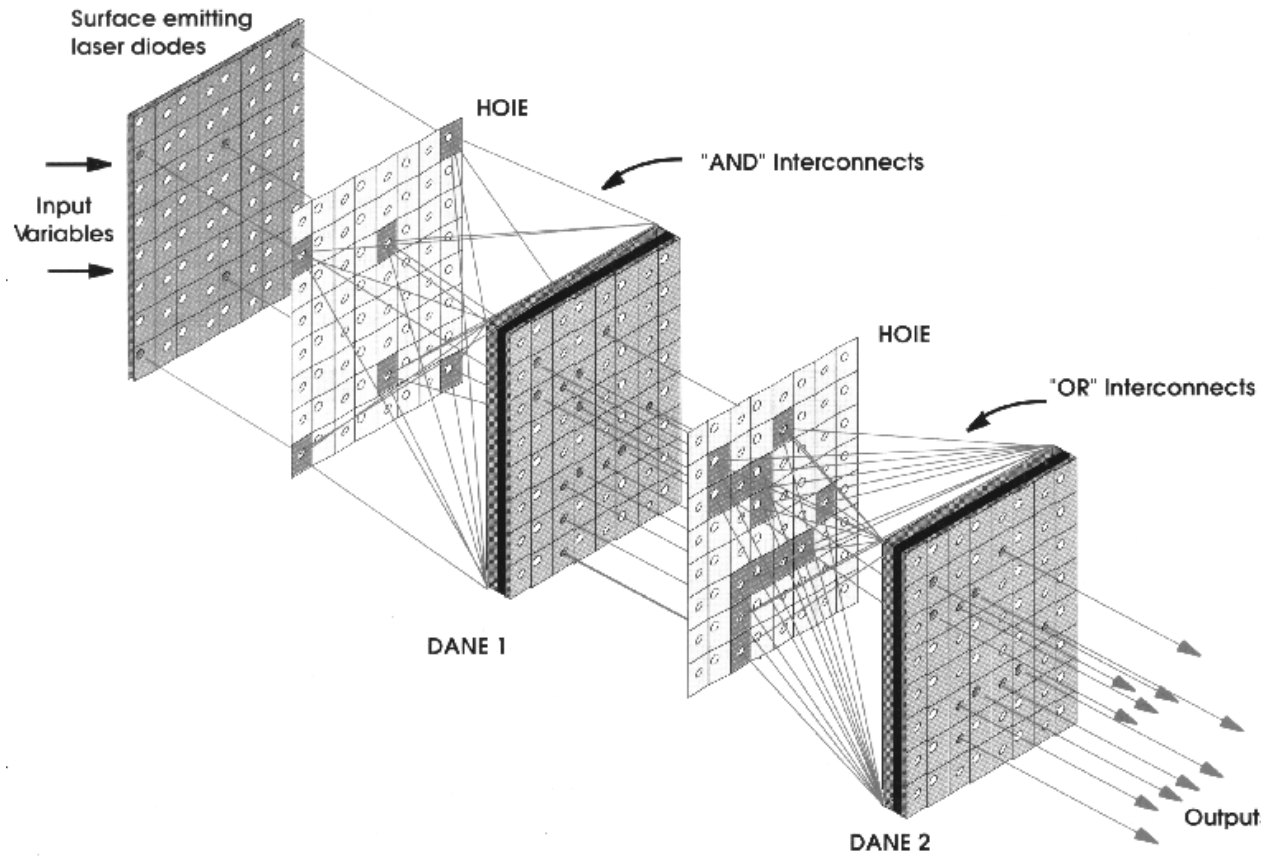
- full digital
- full optical
- hibrid

The principle of the parallel optoelctronic interconnection



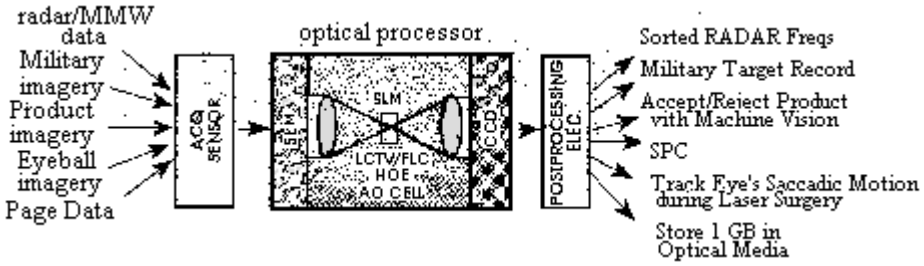
The two planes of spatial light modulators, SLM #1 and SLM #2, modulate in two dimensions the incoming laser light in two-dimensional plane. Each point at plane 1 is connected to its respective point at plane 2 and the digital product A AND B is delivered optically to the photo detection plane or the output.

Global interconnect technology with holographic optical interconnection elements



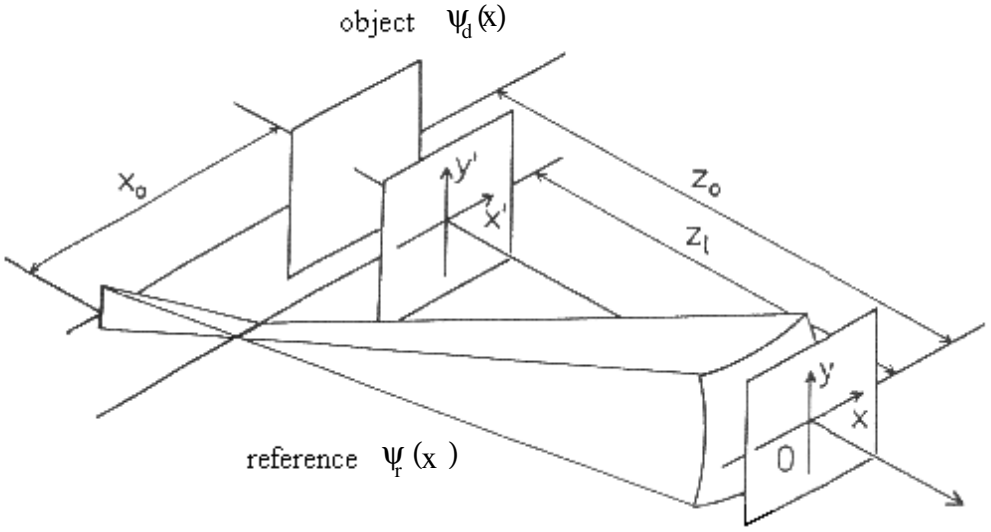
A double-stage high-performance computing module, using arrays of thin planar holographic interconnect elements (HOIEs). (Throughoutput rates on the order of 10^{14} bit operations/second can be achieved with these global interconnect at a 100 MHz clock.) HOIEs selectively image in free space each element on the DANE's (***D***etection, ***A***mplification, ***N***egation and re-***E***mission) laser diode array to predetermined positions and the next DANE detector plane. For this application one can use binary synthetic holograms (HOIEs) maintain space invariance and reconstruction invariance on subsequent DANE planes.

The graphical definition of optical data processor



The acquisition sensor, visible/infrared camera, synthetic aperture radar, spectrophotometer, etc., collects the input information, which is then processed optically to take advantage of the massive parallelism inherent in optics. The results are then converted back into electrons for post processing and displayed to the man/machine interface.

The concept of holograms



hologram $I(x)=|\psi_d(x)+\psi_r(x)|^2$

The intensity distribution in the hologram plane:

$$I(x) = \left|\psi_d(x) + \psi_r(x)\right|^2 = \left|\psi_d(x)\right|^2 + \left|\psi_r(x)\right|^2 + \psi_d(x)\psi_r^*(x) + \psi_d^*(x)\psi_r(x)$$

The reference beam: $\psi_{sph} = \frac{1}{r} e^{\pm ikr}$

$$I(x) = \left| K\exp(ik\alpha_0 x) + A\sum_{n=1}^{\infty} A_n \delta(\xi_n, z_n) \right|^2$$

The hologram function

The positions of the fringes satisfy the following equation:

$$(x/\Delta x) - \left|\Phi(x,y)/2\pi\right| = n$$

In CGH the fringes are binary valued:

$$h(x,y) = \begin{cases} 1, & \cos\Phi(x,y) \geq \cos\left[\pi \sin^{-1}(A(x,y)/\pi)\right] \\ 0, & \text{otherwise} \end{cases}$$

The transmittance of the hologram can be represented as:

$$h(x,y) = \sum_{m=-\infty}^{\infty} \frac{\sin\left[m\pi \sin^{-1}\left(A(x,y)/\pi\right)\right]}{m\pi} \exp\left[im\Phi(x,y)\right]$$

By means of residue arithmetic we obtain the following inequalities:

$$Mod\left[k_x + \frac{M\Phi\left(Tk_x/M, 2Tk_y\right)}{2\pi}\right] \leq Mod\left[\frac{Mq}{2}\right]$$

and

$$Mod\left[k_x + \frac{M\Phi\left(Tk_x/M, 2Tk_y\right)}{2\pi}\right] \geq Mod\left[M - \frac{Mq}{2}\right]$$

where Tk_x/M is substituted for x and $2Tk_y$ for y and the accuracy of the fringe positions can be set by the integer M.

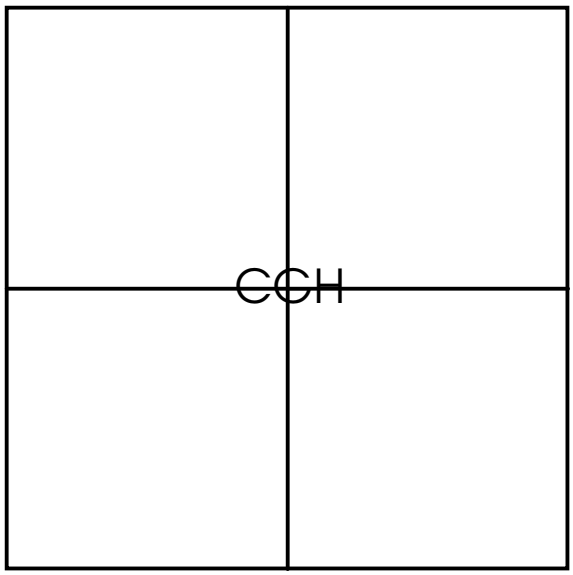
The algorithm

The equation for determining the fringe location to make the hologram in case of spherical wave is:

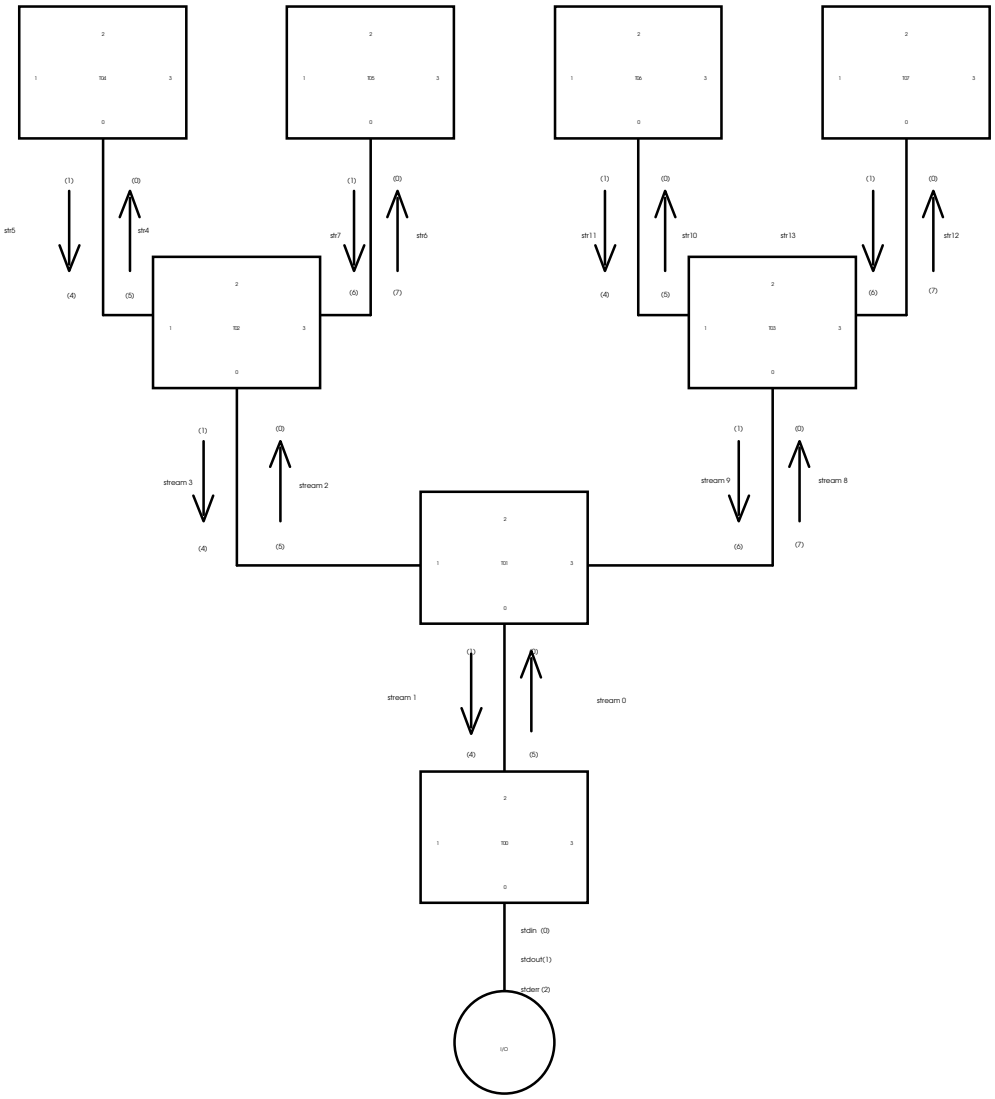
$$Mod\left[k_x + \frac{\left(k_x^2 + 4M^2 k_y^2\right)}{4MN}\right] = 0$$

where $k_x = -MN/2,\ -MN/2-1,\ \ldots,\ MN/2-1,\ MN/2$;
and $k_y = -N/2,\ -N/2-1,\ \ldots,\ N/2-1,\ N/2$.
The parameter N can be interpreted as the number of fringes in the hologram.

Parallelization



The tree structure, the I/O numbers and the data streams



Program module for T00:

```
procedure h00 begin
    welcome the user (stderr)
    short information about the program (stderr)
    get input parameters (stdin)
    initialise/clear variables
    determining the phase space boundaries
    send values to the branch transputer T01
(auxout1)
    do while in legal data space
        collect the matching phase_space points
from T01
                                          (auxin1)
        compute real space point
        set the data matrix
    od
    output the data matrix (stdout)
end of procedure h00.
```

Program module for T01:

```
procedure h01 begin
    read initial data from the root transputer T00
(stdin)
    split up the space in two equally sized pieces
    send values to the branch transputers T02 and
T03
                                (auxout1 and auxout2)
    do while in legal data space
        collect phase_space points from T02 and
T03
                                (auxin1 and auxin2)
        send data to the root T00 (stdout)
    od
end of procedure h01.
```

Program module for T02 and T03:

```
procedure h02 begin
    read initial data from the root transputer T01
(stdin)
    split up the space in two equally sized pieces
    send values to the branch transputers T04 and
T05/T06
                        and T07 (auxout1 and auxout2)
    do while in legal data space
        collect phase_space points from T04 and
T05/T06
                        and T07 (auxin1 and auxin2)
        send data to the root T01 (stdout)
    od
end of procedure h02.
```

Program module for T04...T07:

```
procedure h03 begin
    read initial data from the root transputer
T02/T03 (stdin)
    do while in the required data space
        check phase_space points against the
equations
        if condition fulfilled send data to the
root T02/T03
                                        (stdout)
    od
    set invalid phase_space coordinate indicating
end of job
    send data to the root T02/T03 (stdout)
end of procedure h03.
```

The UNIX standard data streams, extended by the POSIX file descriptor <posix.h> c header file supported by the HELIOS:

Stream	Posix No.	Function
stdin	0	input from the user (T00) or from the root transputer
stdout	1	output to the user or to the root transputer
stderr	2	not redirected, always refers to the console output
auxin1	4	data from the left branch
auxout1	5	data to the left branch
auxin2	6	data from the right branch
auxout2	7	data to the right branch

Network construction

The topology can be specify by a resource map which describes in detail the physical characteristics of the network of processors available to HELIOS. It is used by the network manager, when loading HELIOS onto the network for the first time, and the task force manager, when deciding how to allocate tasks to processors. The resource map file includes the following information:

- the number of processors
- the processor types
- how the processors are interconnected
- how much external memory each processor has

Executing the programs in parallel

The Component Distribution Language (CDL) is used to define groups of related tasks. The definition consists of two parts:

1. a description of the tasks are connected together (i.e. how they communicate each other)
2. an optional description of the resource requirements for each task

The resource map file

```
subnet /Hologram {
CONTROL Rst_Anl (/Hologram/00);
terminal 00{ ~IO,,~01, ;   SYSTEM;
                memory #400000;
                Mnode Rst_Anl(pa_ra.d);
                ptype T800;
        }
terminal 01 { ~00, ~02, ,~03 ;    HELIOS;
                        memory #100000;
                        ptype T800;
        }
terminal 02 { ~01 , ~04, ,~05 ;   HELIOS;
                        memory #100000;
                        ptype T800;
        }
terminal 03 { ~01 , ~06 , ,~07;   HELIOS;
                        memory #100000;
                        ptype T800;
        }
terminal 04 { ~02, , ,  ;     HELIOS;
                        memory #100000;
                        ptype T800;
        }
terminal 05 { ~02, , ,        ;        HELIOS;
                        memory #100000;
                        ptype T800;
        }
terminal 06 { ~03, , ,        ;        HELIOS;
                        memory #100000;
                        ptype T800;
        }
terminal 07 { ~03, , ,        ;        HELIOS;
                        memory #100000;
                        ptype T800;
        }
terminal IO { ; IO; }
}
```

The CDL script:

```
#! /helios/bin/cdl

#
#   Component declarations
#
component h00    {processor T800;
         }
component h01    {processor T800;
         }
component h02    {processor T800;
         }
component h03    {processor T800;
         }
#
#   Task force definition
#
h00 <> h01 (,[2] <> h02 (,[2] <> h03 ))
```

The part of the list produced by CDL:

h00 <> h01 (<> h02 (<> h03 , <> h03) , <> h02 (<> h03 , <> h03))

list of components:

name: h03
code: /Hologram/IO/helios/usr/ghudoba/h03
ptype: T800
puid:
attribs:
memory: 0
streams: <| stream4(0), >| stream5(1), stderr(2) , unused(3)
arguments: h03

name: h03
code: /Hologram/IO/helios/usr/ghudoba/h03
ptype: T800
puid:
attribs:
memory: 0
streams: <| stream6(0), >| stream7(1), stderr(2) , unused(3)
arguments: h03

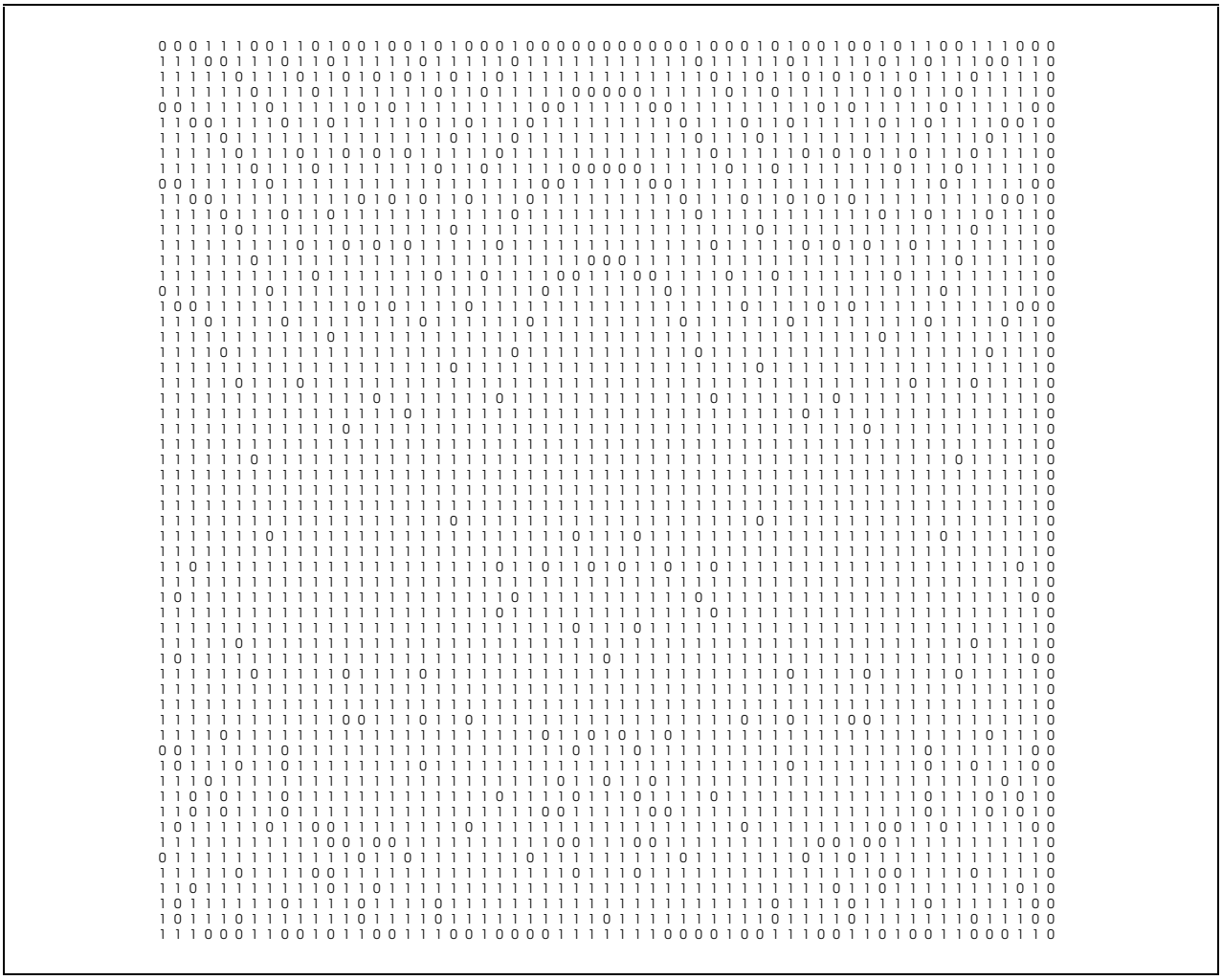
name: h02
code: /Hologram/IO/helios/usr/ghudoba/h02
ptype: T800
puid:
attribs:
memory: 0
streams: <| stream2(0), >| stream3(1), stderr(2) , unused(3) , <| stream5(4), >| stream4(5), <| stream7(6), >| stream6(7)
arguments: h02

*

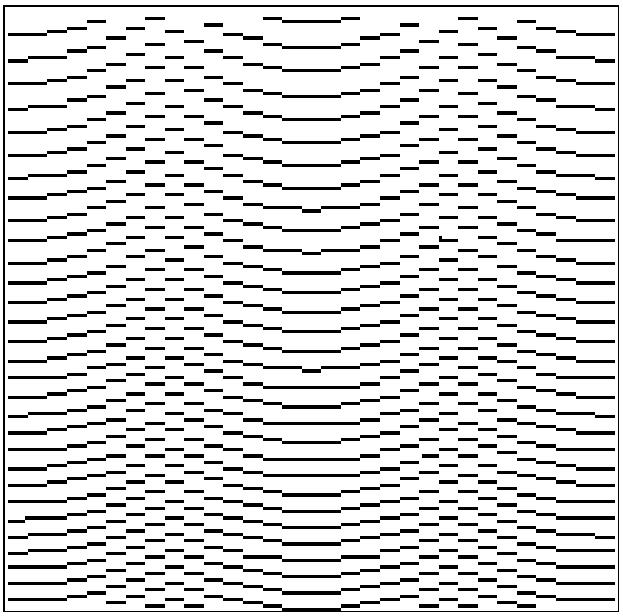
*

*

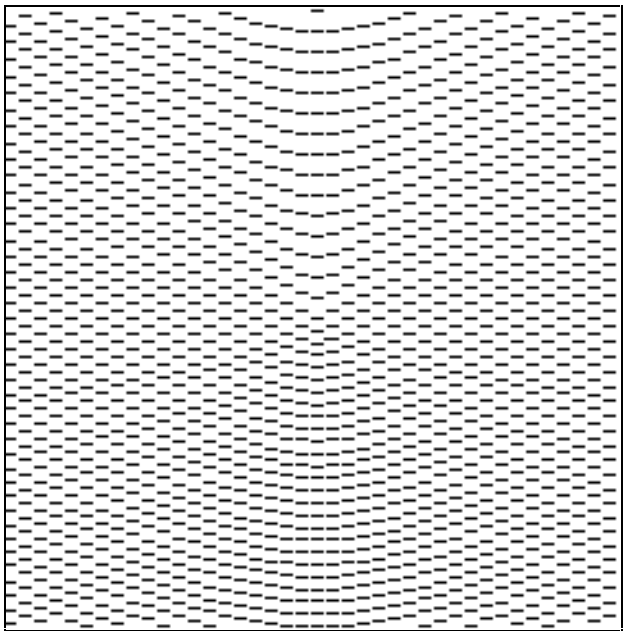
The produced output matrix in case of N=59 and M=99



The computer generated holograms



spherical wave



and conical wave

Van der Pol equation

This nonlinear term containing equation represents an electrical resonant (RLC) circuit in which there is both a non-linear resistance and an active component to overcome the resistance. (The equation has also been used to model pulsating stars called Cepheids.)

$$x''-h(1-x^2)x'+x=0$$

The differential equation above we can solve by a numerical method, using computer. Assuming, that Δt is small enough, so that changes in x and x' is negligible during Δt, we need the following three successive steps:

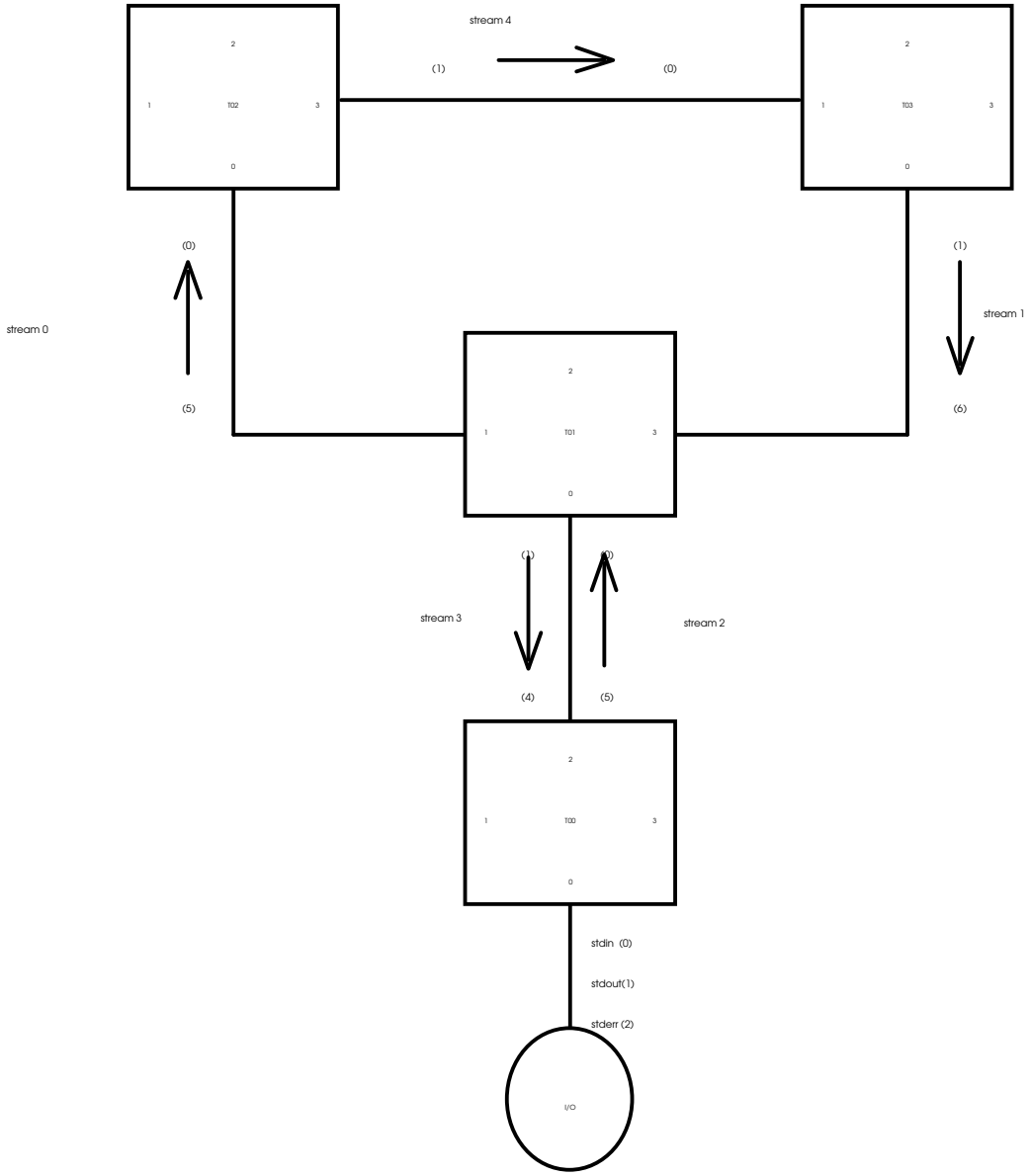
1. $x''_t = h(1-x^2{}_t)\,x'_{t-1/2} - x_t$
2. $x'_{t+1/2} = x'_{t-1/2} + x''_t\,\Delta t$
3. $x_t = x_{t-1} + x'_{t+1/2}\,\Delta t$

The used topology

The definitions file for the ring structure is:

```
subnet /Van_der_Pol {
CONTROL Rst_Anl (/Van_der_Pol/00);
terminal 00{ ~IO,,~01, ;   SYSTEM;
             Mnode Rst_Anl(pa_ra.d);
             ptype T800;
}
terminal 01 { ~00, ~02, ,~03 ;    HELIOS;
                  memory #100000;
                  ptype T800;
      }
terminal 02 { ~01 , , , ~03  ;      HELIOS;
                  memory #100000;
                  ptype T800;
      }
terminal 03 { ~01 , ~02 , ,   ;     HELIOS;
                  memory #100000;
                  ptype T800;
      }
terminal IO { ; IO; }
}
```

The tree structure, the I/O numbers and the data streams



Program module for T00:

```
procedure vp00 begin
    welcome the user (stderr)
    short information about the program (stderr)
    get input parameters: (x,y,h,dt,t) from (stdin)
    compute number of steps: n=t/dt
    initialise/clear variables
    do for i=1 to n
        read from T01 (x,y,z)
        X(i mod Size)=x
        Y(i mod Size)=x'
        Z(i mod Size)=x"
    od
    for i=1 to n do
        write(i,X(i),Y(i),Z(i))
    od
end of procedure vp00
```

Program module for T01:

```
procedure vp01 begin
    get input parameters (x,y,h,dt,n) from (stdin)
    pass data to T02 (x,y,dt,n)
    do for i=1 to n
        z = h(1-x^2)y - x
        write to T02 (z)
        write to T00 (x,y,z)
        read from T03 (x,y)
    od
  end of procedure vp01.
```

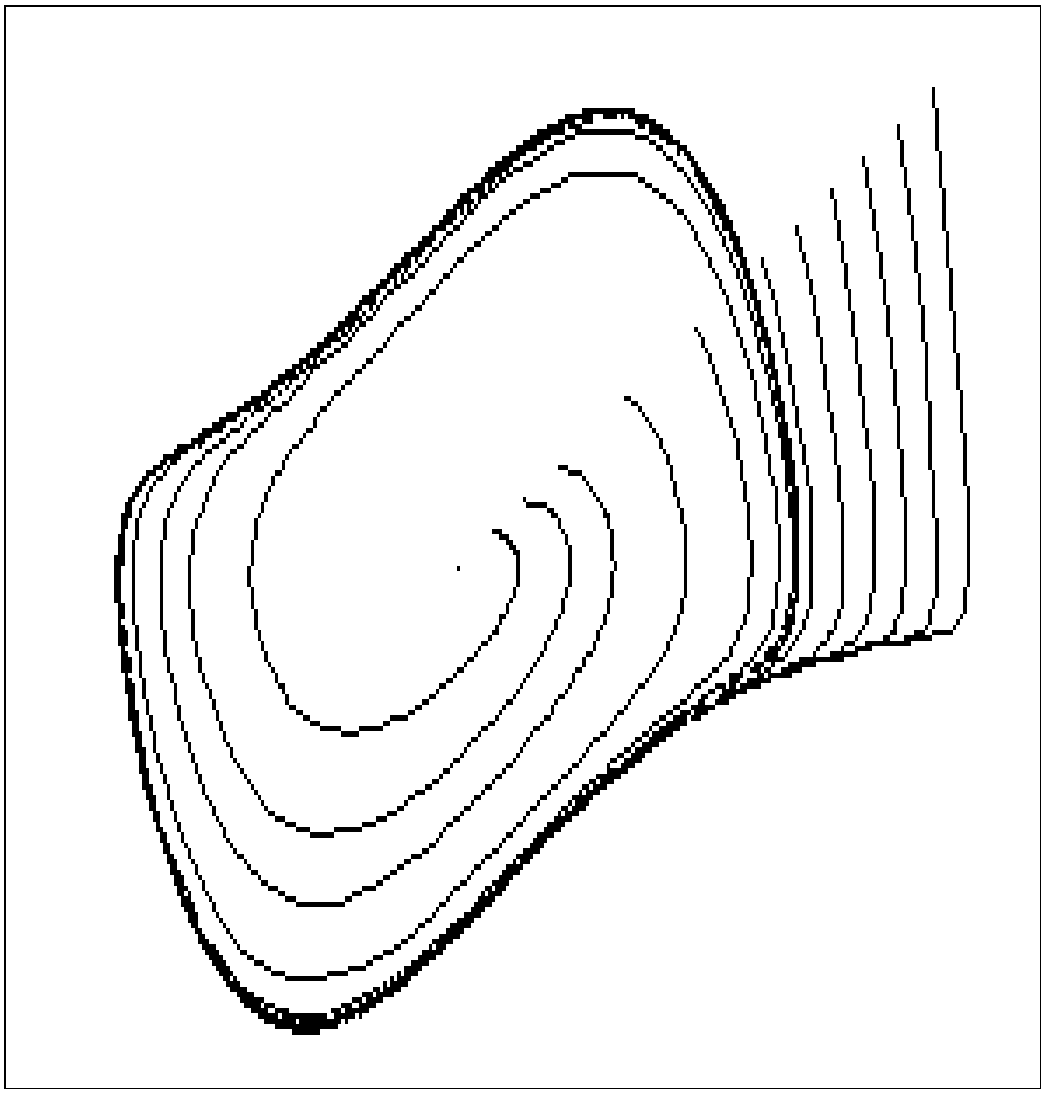
Program module for T02:

```
procedure vp02 begin
    get input parameters (x,y,dt,n) from (stdin)
    pass data to T03 (x,dt,n)
    do for i=1 to n
        read from T01 (z)
        y = y + zdt
        write to T03 (y)
    od
  end of procedure vp02.
```

Program module for T03:

```
procedure vp03 begin
    get input parameters (x,dt,n) from (stdin)
    do for i=1 to n
        read from T02 (y)
        x = x + ydt
        write to T01 (x,y)
    od
  end of procedure vp03.
```

The CDL script:

```
#! /helios/bin/cdl

#
#       Component declarations
#

component vp00 {processor T800;
                }

component vp01 {processor T800;
                puid /Van_der_Pol/01;}

component vp02 {processor T800;
                puid /Van_der_Pol/02;}

component vp03 {processor T800;
                puid /Van_der_Pol/03;}


#
#       Task force definition
#

vp00 <> vp01 <> ( vp02 | vp03 )
```

The graphical plot of velocity versus position starting from different initial conditions