György Hudoba

Exercises on parallel computing

Athens, 13 May – 14 July, 1996.

Introduction

The many transputer containing parallel machines are very useful solving computational hunger problems, having inherent parallelism, such as weather forecast, modelling, image processing, ... and so on. An other application could be to make experiments with different structures, before their have a final hardware realisation. In this paper we describe two worked examples from the field of signal processing, illustrating both types of problems mentioned above. The examples are keeping single and stupid in order to help understand how to analyse and parallelize the problem as well as how to work with the Parsitec HELIOS parallel operating system. The programs were written in **'c'** programming language.

1. Parallel computing of binary holographic optical elements

The digital optical computing

Before we deal with the programming and the program itself, first we give a short explanation of the digital optical computing, than we show how to generate a binary synthetic holograms, which are used in the optical computing units.

Holographic optical elements (HOEs), or diffractive optics, are of wide interest due to their many applications a. g. in integrated optics, in optical data and image processing, and so on. HOEs have several advantages compared to conventional optical elements. They are lighter, more compact, and can be formed and replicated with relative ease. Even more important is the fact, that they can sometimes perform complex optical operations which cannot be achieved with conventional way.

As early as the 1960s researchers could see the potential benefit of optical computing. (The optical correlation was first demonstrated by Anthony VanderLugt in 1963.) Operating at the speed of light and without electromagnetic interference, an optical computer promised to solve myriad data- and image-processing problems from medicine to the military. In the past two to three years the research is focused into two application-driven thrust: optical interconnects in digital systems and optical signal and image processing.
One bottleneck in modern parallel computers is in communicating information between circuit boards. Electronic methods are limited in channel bandwidth, dissipate more power and suffer from electromagnetic interference. The optical interconnections in digital systems support fast photonic interconnections, especially the use of lasers to beam tightly aimed light signals directly from one board to another in so-called free-space optical interconnects. The

interconnection sites take the form of smart-pixel arrays, in which each element of the array contains a photo detector, an optical source or modulator, and some electronic logic circuitry.

The latest work has involved four technologies that enable free-space interconnects between smart-pixel arrays. These technologies are:

- Vertical-cavity surface-emitting laser (VCSEL) arrays for transmission.
- High-speed metal-semiconductor-metal (MSM) photo detector arrays on silicon membranes integrated with complementary metal-oxide semiconductor (CMOS) electronics for detection.
- Flip-chip bonding for combining the transmission and reception optoelectronics.
- Holographic optical interconnection elements to direct the free-space laser beams.

The first three technologies combine to produce the major building block for free-space interconnects between high-speed smart-pixel arrays. The final technology directs beams by space light modulators (SLMs) from one array to the appropriate photo detectors on other arrays, thereby achieving a board-to-board interconnection several times faster and more dense than an electrical interconnection.

In a parallel configuration (Fig. 1), two planes of spatial light modulators, SLM1 and SLM2, 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. This is referred to as parallel interconnect, because all interconnects are in fact parallel with respect to each other.

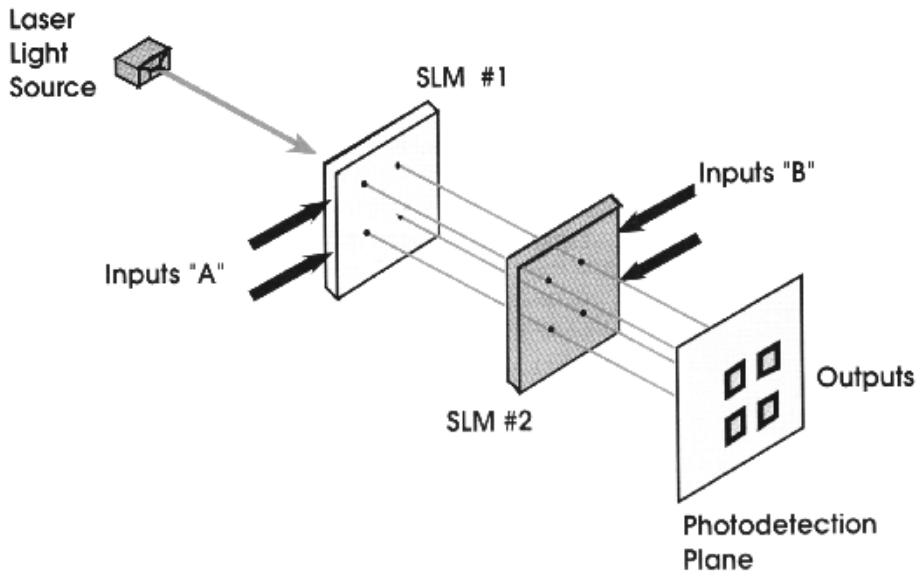

Figure 1. The principle of the parallel optoelctronic interconnects

The second generation digital optical computer (DOCII) architecture is configured as a matrix/vector Boolean logic multiplier (Fig. 2.). Each column of the two-dimensional control operator is bit-wise multiplied against the input data vector. These products are Boolean summed on their respective output detectors.

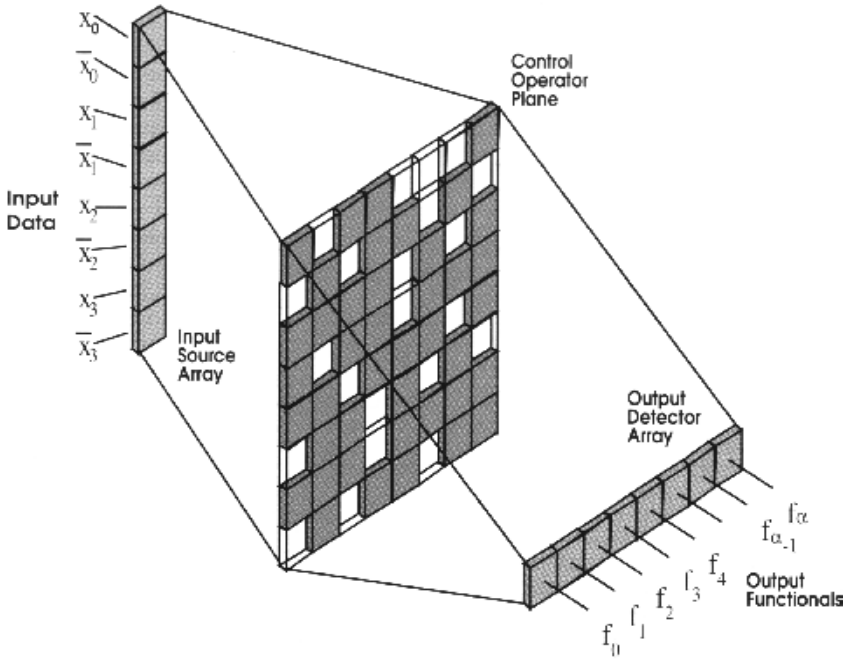

Figure 2. Reconfigurable, programmable logic array for DOC II.

With the global interconnect in Figure 3, two planes of modulators modulate in two dimensions the incoming laser light in a two-dimensional plane, but the interconnects between two planes are now global. Each point in plane 1 is selectively broadcast to all possible modulators at the second plane. The digital products A_m AND B_n is selectively broadcast to all possible detectors by optical delivery to the photo detection plane or the output. This is referred to as global interconnect because all points are globally broadcast to all other points at each successive optical modulation plane or detection plane.

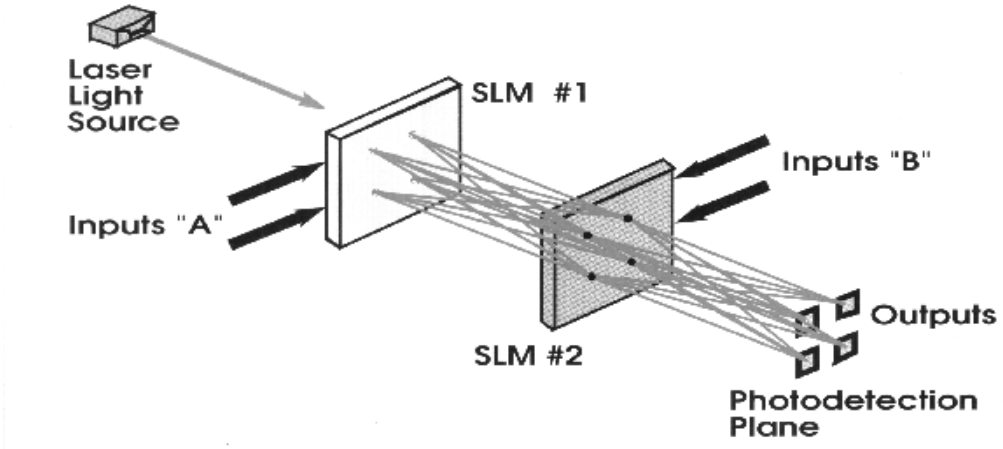

Figure 3. Global optoelectronic interconnects

The next-generation DOCIII hardware uses a global interconnect topology. As shown in Figure 4, a double-stage high-performance computing module, interconnects between the input and the subsequent stages are globally broadcast by an array 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 (detection,

amplification, negation and re-emission) laser diode array to predetermined positions and the next DANE detector plane. For this application one can use binary synthetic holograms (HOIEs, which is the aim of our work) to maintain space invariance and reconstruction invariance on subsequent DANE planes.

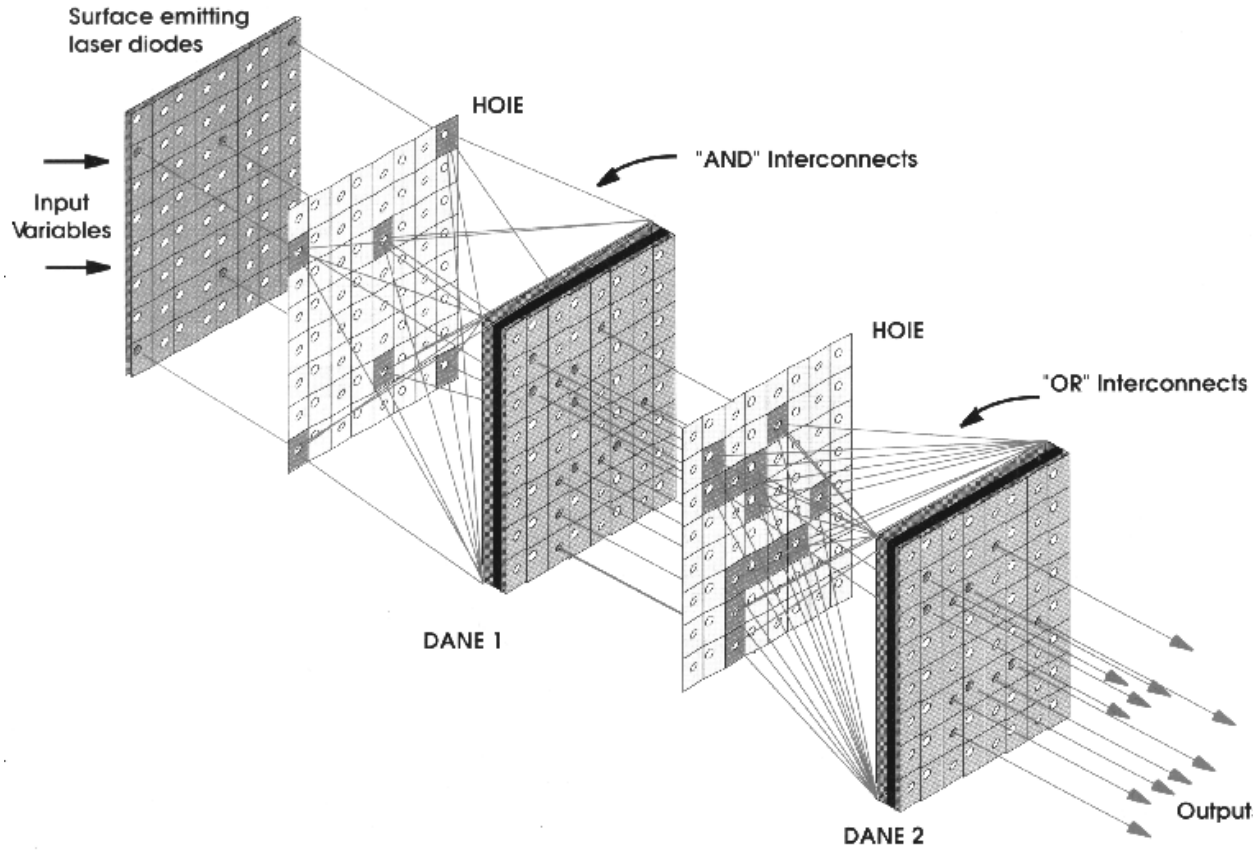

Figure 4. Global interconnect technology with holographic optical interconnect elements

Figure 5. is a graphical definition of optical processing. 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. When an optical correlator processes a scene using the inherent Fourier-transform properties of lenses, it optically compares all pixels in the scene to all pixels in the reference image simultaneously. (This parallel processing is the key feature that distinguishes optical correlators from their digital correlator counterparts.) A "match", or correlation, between the reference object and the input object shows up as a focused spot of light on an otherwise black background, like a star in the night sky. Objects, that match perfectly produce an ideal response, a sharp, bright point.

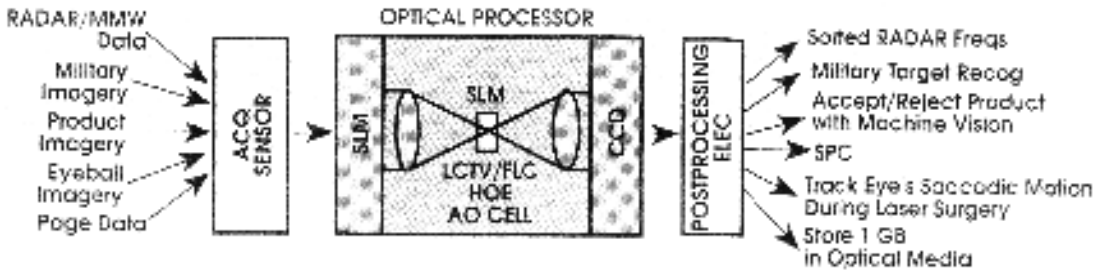

Figure 5. The optical data processor

Not only does a correlator recognise object, it locates them at the same time. An important property of optical correlation is that the correlation spot will not shift in position even when portions of the recognised object are hidden from view. The intensity and position of a correlation spot is independent of the features in the rest of the scene. Thus, the output of a correlator is map of values representing the likelihood that an object or group of features at a particular location in the scene is on of the interest.

The third important property of optical correlators is that when more than one object lies in the input scan, the correlator processes them all at once, with individual accept/reject indications for each object. Each sub-images produces its own correlation spot, properly summed and properly located, even if there are competing features in the input such as background clutter. The result is parallel processing of multiple objects, in addition to parallel processing of the pixels.

The power of parallelism enables optical correlators to operate on an enormous amount of data, such as photograph. A 35mm film frame contains roughly 30 million data points, each up to 7 bits deep, or about 25 Mbyte information total. In fact, since light is used, the optics processes the image with a speed of light (3000 000 km/s) in about 2 ns. However the system's real cycle time is significantly slower, because of the serving electronics.

The concept of holograms

Information about holography first appeared in published form in the late 1940s in three classic papers written by Denis Gabor. He has worked out a method, how to record not only the intensity distribution of an object, but the phase of the incoming waves too, resulting the 3D image of the objects. Subsequent work involving the use of an off-axis reference wave, published in the early 1960s by Leith and Upatnieks at the University of Michigan, coupled with the invention of the laser, revitalised interest in the subject and led to a large amount of research.

A hologram is nothing else than an interferogram. In essence, this means, that the hologram is made in such a manner, that the phase information associated with the diffraction pattern of the object is preserved during the recording step of the process. Thus a hologram is a recording of the intensity distribution describing the spatial interference between an object field $\psi_d(x)$, (a Fresnel or Fraunhofer diffraction pattern arising from transmission through or reflection from an object) and a reference field $\psi_r(x)$ (plane or spherical wave either coaxial with or making an angle to the optical axis):

$$I(x) = \left|\psi_d(x) + \psi_r(x)\right|^2 = \left|\psi_d(x)\right|^2 + \left|\psi_r(x)\right|^2 + \psi_d(x)\psi_r^*(x) + \psi_d^*(x)\psi_r(x) \qquad (1)$$

A spherical wave may be expressed mathematically by the equation $\psi_{sph} = \frac{1}{r} e^{\pm ikr}$, where *r* is the distance from the point source to the point of observation, and the ± indicates a diverging and a converging wave respectively.

Holograms can be classified in many ways:

- *Fresnel hologram* – if the hologram is in the Fresnel diffraction region of the object, and the reference field is coaxial with the illumination field
- *Fraunhofer hologram* – if the hologram is in the Fraunhofer diffraction region of the object, and the reference field is coaxial with the illumination field
- *Sideband Fresnel/Fraunhofer hologram* – if the hologram is in the Fresnel/Fraunhofer diffraction region of the object, and the reference field is makes an angle with respect to the optical axis
- *Fourier transform hologram* – if the hologram intensity of Eq. 1. is of the form $I(x) = \left| K + \overline{D}(x) \right|^2$, where $\overline{D}(x)$ denotes an exact Fourier transform of the object.

To reconstruct an image from the hologram, the hologram is placed in a coherent optical beam. The image then formed at a position determined from the type of hologram that was produced.

In order to compute the intensity distribution in the hologram plane, we consider the three-dimensional object as a set of individual points, and express it as:

$$O(\xi, z) = \sum_{n=1}^{\infty} A_n \delta(\xi_n, z_n) \tag{2}$$

where A_n (the brightness of the body at the given point) are constants. Then the intensity distribution in the hologram is given by:

$$I(x) = \left| K \exp(ik\alpha_0 x) + A \sum_{n=1}^{\infty} A_n \delta(\xi_n, z_n) \right|^2 \tag{3}$$

where A and K are constants. The definition of a 3D object and the ray tracing is far beyond of this paper. (For the details see e.g. Ref. 4.)

As the matter of fact the hologram is noting else, than a series of fringes, formed by interfering a wavefront coming from each point of the object with a usually tilted plane (or sometimes spherical) reference wave. The positions of the fringe satisfy the following equation:

$$(x/\Delta x) - \left| \Phi(x,y)/2\pi \right| = n \tag{4}$$

where *n* are integers. In the absence of any phase variation due to $\Phi(x,y)$ the hologram thus formed is a periodic grating with period Δx. It can be seen,

that the effect of the phase function is to modulate the position of the fringe.
In an optically produced holograms the fringes have grey levels. As one scans across the hologram, the transmittance of the photographic medium varies in a gradual fashion from transparent to opaque as one moves across the fringes. However, with a CGH there are no grey levels, because computer controlled plotting devices usually have binary-valued outputs. The result being that in CGH the fringes are binary valued. So, across the hologram the transmittance is either 0 or 1. How do we decide to plot a zero or a one? To do this we need to consider the wavefront that we want to produce with the hologram. Suppose we denote this wavefront by $A(x,y)\exp(\Phi(x,y))$. At some point (x,y) we plot either a 0 or 1. We do this as follows:

$$h(x,y)=\begin{cases}1, & \cos\Phi(x,y)\geq\cos\left[\pi\sin^{-1}(A(x,y)/\pi)\right]\\ 0, & \text{otherwise}\end{cases} \tag{5}$$

where the maximum value of $A(x,y)$ is normalised to 1. So, value of $h(x,y)$ is either 0 or 1. Without going into the details of the derivation, it can be shown, that the transmittance of the hologram can be represented as:

$$h(x,y)=\sum_{m=-\infty}^{\infty}\frac{\sin\left[m\pi\sin^{-1}\left(A(x,y)/\pi\right)\right]}{m\pi}\exp\left[im\Phi(x,y)\right] \tag{6}$$

The summation is over the various diffraction order. Note, that the m=1 diffraction order reproduces the desired wavefront. However as the wavefronts are shown here, the multiple diffraction orders will interfere with one another. This can be corrected by adding a tilt to the phase term $\Phi(x,y)$. In particular, we replace $\Phi(x,y)$ by $\Phi(x,y)+x/T$. This phase tilt causes the diffracted orders to separate in space as they propagate away from the CGH.

Computing binary synthetic holograms

The general problem of making binary synthetic hologram is to find a hologram function $h(x,y)$ that has 0 or 1 as its values and can produce any desired wavefront just by changing its parameters. As it can be see, computing holograms is very processor hungry, so a number of methods and tricks were developed in order to reducing the computation time. After all (the details see in Ref. 5.) by means of residue arithmetic we obtain the following inequalities:

$$Mod\left[k_x+\frac{M\Phi\left(Tk_x/M,2Tk_y\right)}{2\pi}\right]\leq Mod\left[\frac{Mq}{2}\right] \tag{7}$$

and

$$Mod\left[k_x+\frac{M\Phi\left(Tk_x/M,2Tk_y\right)}{2\pi}\right]\geq Mod\left[M-\frac{Mq}{2}\right] \tag{8}$$

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 binary synthetic hologram can be made by testing each point (k_x,k_y) against inequalities (7) and (8). This greatly simplifies the computer programming involved in solving our problem. (In the following for (k_x,k_y) we refer as phase space point, for (x,y) as real space or hologram point.)

If the point satisfies either one of the inequalities, a line segment of length 2T will be drawn parallel to the *y* axis about that point. It is also possible to find the locations of all the points that belong to the same fringe and connect the points to make up the different fringes in the hologram. The amount of computation involved is proportional to the number of points in the hologram plane. If a hologram has N fringes, the amount of computation is roughly equal to M(N)2-2.

The final algorithm

In our worked example we compute hololograms for spherical and for conical wavefronts.

1. Spherical wave: $\Phi(x,y)=\pi\left(x^2+y^2\right)/\lambda F$

Here *F* is the focal length and λ is the wavelength of the coherent light. The equation for determining the fringe location to make the hologram is:

$$Mod\left[k_x+\frac{\left(k_x^2+4M^2k_y^2\right)}{4MN}\right]=0 \qquad (9)$$

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.

2. Conical wave: $\Phi(x,y)=2\pi\sqrt{\left(x^2+y^2\right)}/r_0$

$$Mod\left\{k_x+\frac{\sqrt{\left(k_x^2+4M^2k_y^2\right)}}{4}\right\}=0 \qquad (10)$$

Parallelization

The next step is examining the program in detail to see how it can be split up into tasks which can run in parallel. In our case the intensity value in each

point of the hologram can be calculated independently so in principle the program can be split as many parallel processes, as many points of the hologram has. It seems to be silly, because we would need enormous amount of processors, and the communications overhead would be incredible high. The so called "coarse-grain parallelism" is much more efficient.

In our example we split up the hologram area to four equally sized pieces, and the application specific topology is a tree. Each branch processor on the same level in the tree will process the same algorithm, called h00, h01, h02 and h03. The matching points according to equ. (9) and (10) is computed by transputer T04, ...T07 according to the loaded program h03. They give the data to their roots, T02 and T03 respectively. These transputers (the loaded program is h02) collect the data and forward them to their root T01 (h02). The

result is then forwarded to T00. This processor from the obtained (k_x,k_y) phase points calculates the *(x,y)* real space points coordinates of the hologram to print or file the matrix containing 0-s and 1-s. From this file we can draw the hologram, which after photo reduction is ready to use in an optical device.

In the Figure 6. we show the tree, the used transputers, numbered from **T00** to **T07**, the links of the processors form **0** to **3** in each transputer box, the data streams from **stream 0** to **stream 13** allocated by the **HELIOS** *parallel operating system*, and the I/O numbers in braces. The UNIX standard data streams are 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

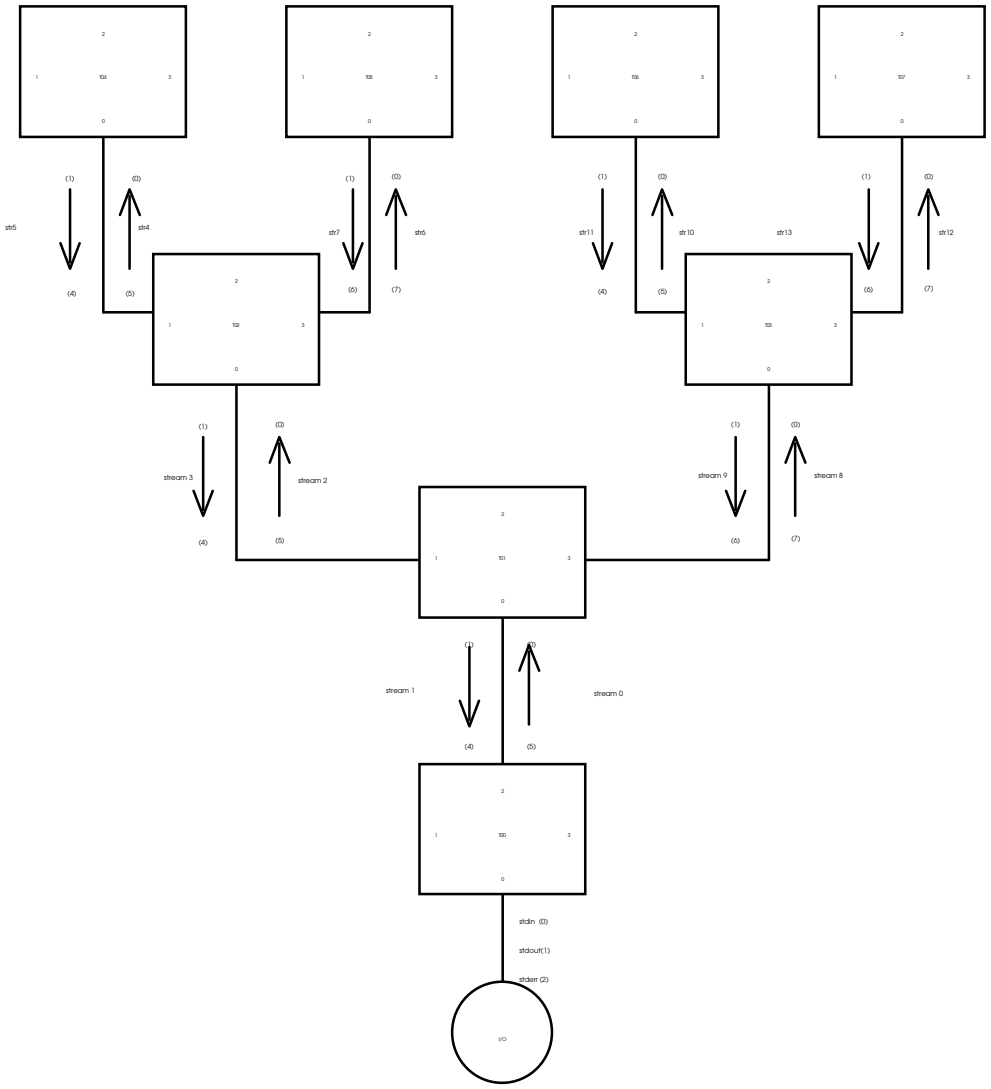

Figure 6. The tree structure, the I/O numbers and the data streams

The program modules

The code of the following program modules are written in HELIOS **c** programming language, which is an ANSI **c** extended with parallel programming tools..

1. 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.
```

2. 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.
```

3. 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.
```

4. 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 equation (9) or (10)
                  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.
```

Network construction and executing the programs in parallel

The next problem is to constructing the desired application specific topology. At the booting process the system loads the default.map file which is defines the transputer structure and networks the transputers according to the user's need. The topology can be specify by a resource map. The resource map file 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

The resource map file must be compiled by the utility program rmgen before it can be used by the network manager. The program simply converts the textual resource map file into a more efficient binary representation. In our case the hologram.rm resource map file is the following:

```
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; }
}
```

To compile this file we should type: rmgen hologram.rm -o default.map

In the following we give a short explanation of the resource map. The subnet/Hologram defines a new subnetwork. The single processor in the network is defined with the terminal construct. The curly braces after the processor name contain the processor definition. This consists of two parts:

1. a definition of how the processor's links are connected to other processors in the network
2. a series of optional statements

The link description part,'~IO, , , ;' has four fields separated by commas, one for each of the processor links numbered from 0 to 3 in the Fig. 6. The first field corresponds to link 0, the second to the 1 and so on. The details can be found in Ref. 8.

The next problem is to express this system in terms of the Component Distribution Language (CDL). The 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 CDL task force description is used by the task force manager when loading task force onto the network. The task force manager attempts to reconcile the resource requirements as specified in the CDL file, with the network resources as specified in the resource mapfile described above. The task force manager also ensures that the tasks are correctly connected together. The file which reflect our tree structured solution is the following:

```
#! /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 explanation of the command lines is can be found in Ref. 9. This CDL script can be run using the command line:

```
cdl hologram.cdl
```

If we want check the created task force, we can produce a list file by the command

```
cdl -l holocdl.l hologram.cdl
```

The results

If we run our program we can get an output matrix in case of N=59 and M=99. The result is presented here. From this matrix can be construct the hologram itself, which is also presented, although it is beyond of the scope of this paper.

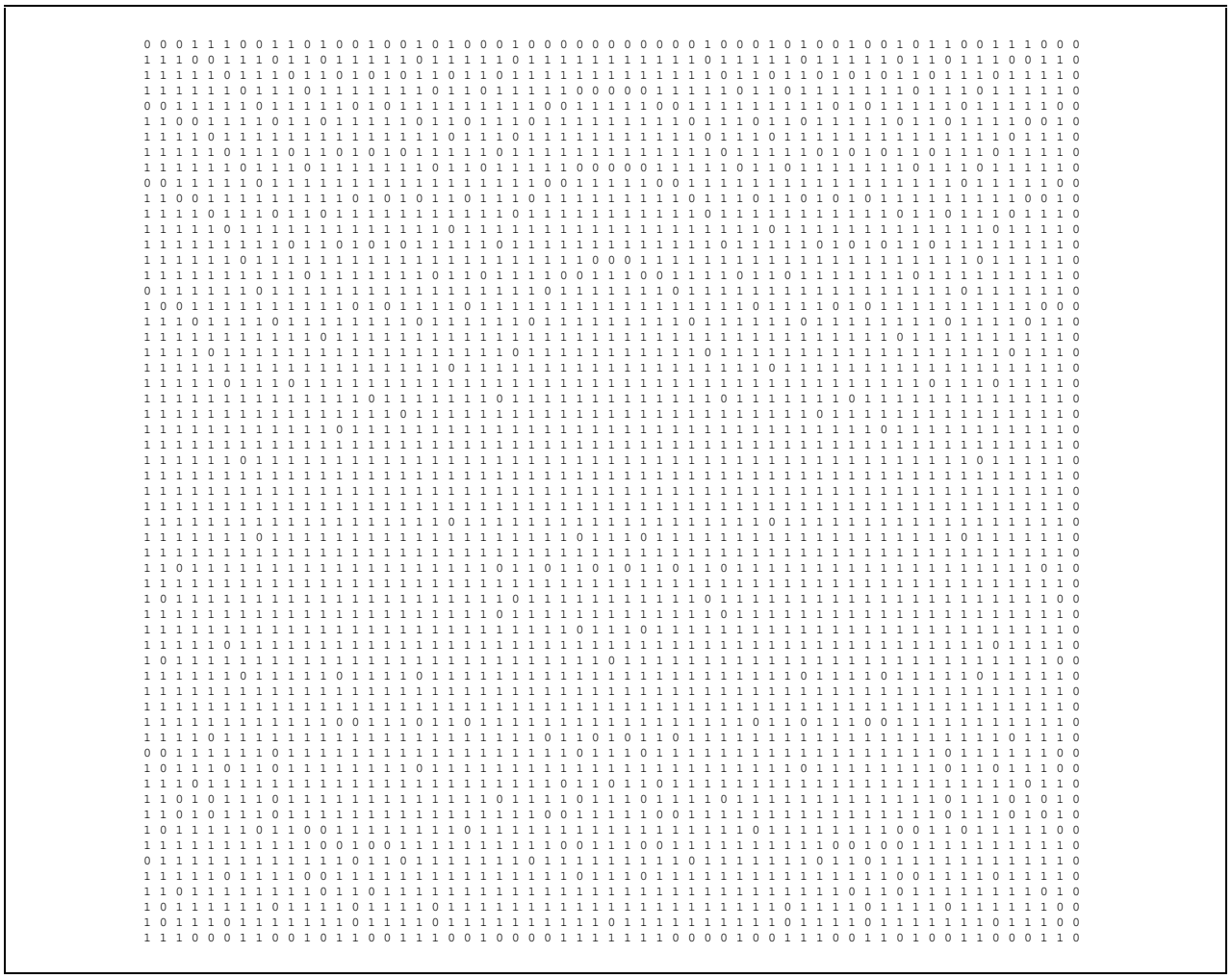

Figure 7. The data matrix for spherical wave

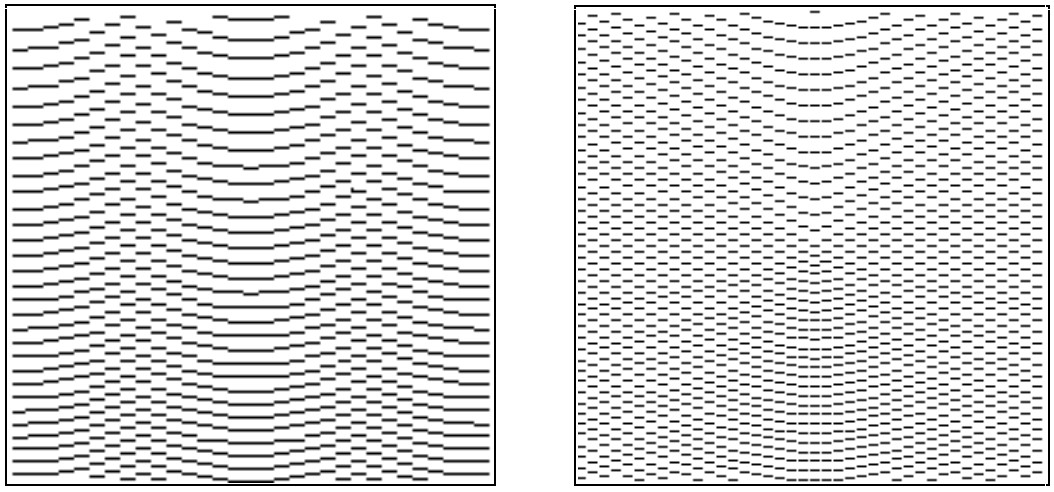

Figure 8. The computer generated hologram in case of spherical and conical wave

György Hudoba

Exercises on parallel computing

Athens, 13 May – 14 July, 1996.

Modelling of a non-linear oscillator (Van der Pol equation)

Many non-linear systems leads to irregular and unpredictable time evolution. It occurs in mechanical oscillators such as pendulum or vibrating objects, in rotating or heated fluids, in laser cavities, in some chemical reactions, in the atmosphere or in the electronics. Its central characteristic is that the system does not repeat its past behaviour (even approximately). Yet, despite their lack of regularity, the so called chaotic dynamical systems follow deterministic equations such as those derived from Newton's second law, or Maxwell's law, ... and so on.

The unique character of chaotic dynamics may be seen most clearly by imagining the system to be started twice, but from slightly different initial conditions. We can think of this small initial difference as resulting from measurement error, for example. For nonchaotic systems this uncertainty leads to an error in prediction that grows linearly with time. For chaotic system, on the other hand, the error grows exponentially in time, so that the state of the system is essentially unknown after a very short time.

Chaotic motion is not a rare phenomenon. Consider a dynamical system described by a set of first order differential equations. If the system has at least three independent dynamical variables, or the equations of motion contain a non-linear term, that couples several of the variables, the behaviour of the system could be chaotic.

Such a nonlinear term containing equation was first used by Van der Pol to represent 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.) The Van der Pol equation, given by:

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

For h=0, the equation is just that of an undamped harmonic oscillator that has simple periodic orbits all of the same frequency. For h>0, large oscillations are damped and small oscillations grow as a result of the non-linear damping term. If the results are plotted in phase space (velocity versus position), all orbits are attracted to a closed curve, which is called a limit cycle or attractor.

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. $x_t = x_{t-1} + x'_{t+1/2} \Delta t$

Parallelisation

If we use a three transputer containing ring structure (Fig. 9.), each processor computes only one step, and forwards the result to the next. In this case the speed of the computation is increased by the factor of three. Analysing the step 1. we can find an inherent parallelism.

1.1 $A = x^2$

1.2 $B = 1 - A$

1.3 $C = Bhx'$

1.4 $D = C - x$

In step 1.3 the product hx' can be calculated independently. The four steps we could reduce to three, containing in each step only one multiplication process.

1.1 $A = x^2$

1.2 $B = 1 - A$ and $C = hx'$ in parallel

1.3 $D = BC - x$

According to this analysis we could compute the step 1 again a 3 transputer ring network. This consideration leads us to the so called fine-grain parallelism. But this requires significantly more programming effort, and perhaps the application goes slower when implemented in full parallel because the communication overhead between tasks is greater than the time each tasks takes processing data.

The last few numerical values are stored in three vectors X, Y and Z. We use them as windows, not to overrun the subscripts. This is the reason for the modulo arithmetic in module vp00.

The used topology

The definitions file for the ring structure is the following:

```
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; }
}
```

In the Figure 9. we show the ring, the used transputers, numbered from **T00** to **T03**, the links of the processors from **0** to **3** in each transputer box, the data streams from **stream 0** to **stream 4** allocated by the **HELIOS** *parallel operating system*, and the I/O numbers in braces.

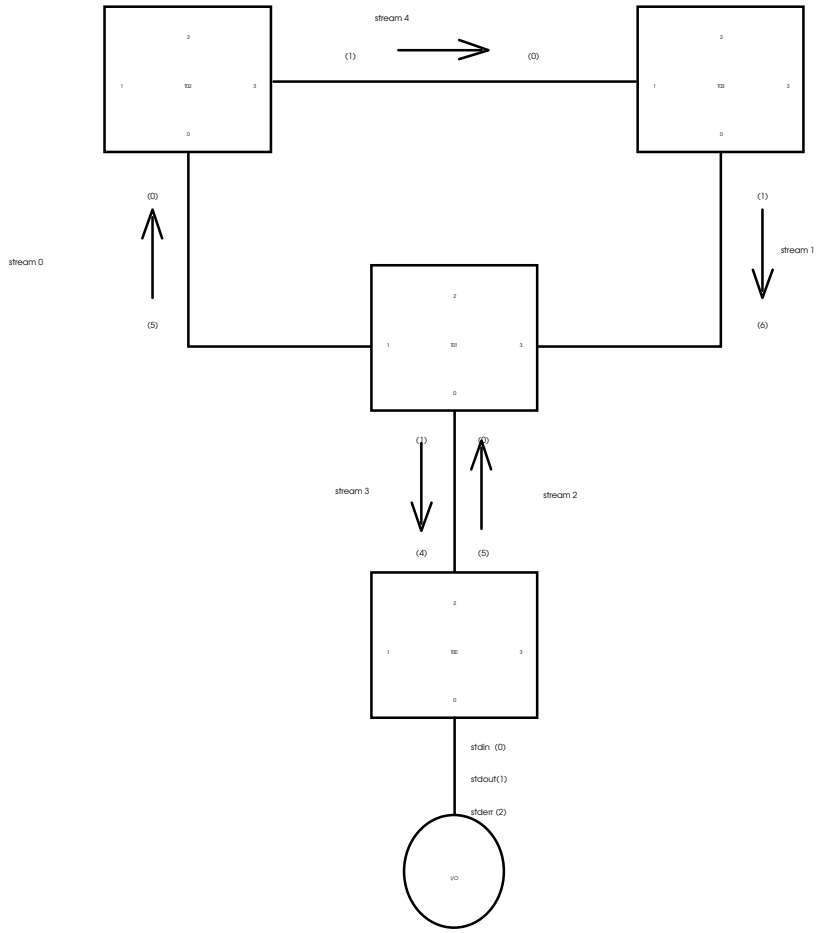

Figure 9. The ring structure, the I/O numbers and the data streams

The program modules

The four program modules:

1. 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
```

2. 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.
```

3. 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.
```

4. 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

In order to run the program in parallel, we used the following CDL definitions file:

```
    #! /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 next step is to let run the program. The output as a result looks like:

```
X=  1.000000 Y=  0.000000 Z= -1.000000
X=  0.960000 Y= -0.200000 Z= -0.988224
X=  0.880471 Y= -0.397645 Z= -1.041353
X=  0.759288 Y= -0.605915 Z= -1.221157
X=  0.589259 Y= -0.850147 Z= -1.588176
      :
and so on
```

Using different initial values we can examine the behaviour of the oscillator.

Acknowledgements

This work was sponsored by the TEMPUS. The 16 transputer containing PARSITEC and the HELIOS parallel programming tool was provided by the University of Athens. Special thanks to Prof. Sergios Theodoridis, dr. Yianis Cotronis and Costas Vassilakis.

References

1. Guilfoyle, P.S.: A third-generation digital optical computer, *Photonics Spectra* (May 1993)

2. Bagley, H. and Sloan, J. :Optical processing: Ready for machine vision? *Photonics Spectra* (August 1993)

3. Johnson K. and Neff, J.: Improving medicine, manufacturing and the military *Photonics Spectra* (November 1995)

4. HP Journal, October, 1995.

5. Wai-Hon Lee: Binary synthetic holograms Applied optics (July 1974, vol. 13. no 7)

6. Cindirch, I.N. and Lee, S.H.: Holographic Optics:Optically and Computer Generated, SPIE Proc. v.1052.

7 Reynolds, DeVelis, Parrent, Thompson: The new Physical Optics Notebook: Tutorials in Fourier Optics

8 The HELIOS Parallel Programming Tutorial

9 The CDL Guide