TEMPUS: Activity 2

PARALLEL ALGORITHMS

Chapter 4

Signal and Image Processing Algorithms

István Rényi, KFKI-MSZKI

1 Introduction

Before engaging in spec. purpose array processor architecture and implementation, the ***properties and classifications of algorithms*** must be understood.

Algorithm ***is a set of rules for solving a problem in a finite number of steps***

- Matrix Operations
- Basic DSP Operations
- Image Processing Algorithms
- Others (searching, geometrical, polynomial, etc. algorithms)

1 Introduction - continued

Two important aspects of algorithmic study:

application domains and ***computation counts***

Examples:

Application domains

Application	Attractive Problem Formulation	Candidate Solutions
Hi-res direction finding	Symmetric eigensystem	SVD
State estimation	Kalman filter	Recursive least squares
Adaptive noise cancellation	Constrained last squares	Triangular or orthog. decomposition

1 Introduction - continued

Computation counts

Order	Name	Examples
N	Scalar	Inner product, IIR filter
N^2	Vector	Lin. transforms, Fourier transform, convolution, correlation, matrix-vector products
N^3	Matrix	Matrix-matrix products, matrix decoposition, solution of eigensystems, least square problems

Large amounts of data + tremendous computation requirement, increasing demands of speed and performance in DSP =>

=> need for ***revolutionary supercomputing technology***

Usually multiple operations are performed on a single data item on a *recursive* and *regular* manner.

2 Matrix Algorithms

Basic Matrix Operations

- Inner product

$$\mathbf{u}^T = [u_1, u_2, \ldots, u_n] \quad \text{and} \quad \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ . \\ . \\ . \\ v_n \end{bmatrix}$$

$$\langle \mathbf{u}, \mathbf{v} \rangle = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n = \sum_{j=1}^{n} u_j v_j$$

2 Matrix Algorithms - continued

- Outer product

$$\begin{bmatrix} u_1 \\ u_2 \\ . \\ . \\ . \\ u_n \end{bmatrix} \cdot \begin{bmatrix} v_1 & v_2 & \dots & v_m \end{bmatrix} = \begin{bmatrix} u_1v_1 & u_1v_2 & \cdots & u_1v_m \\ u_2v_1 & u_2v_2 & \cdots & u_2v_m \\ u_3v_1 & u_3v_2 & \cdots & u_3v_m \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ u_nv_1 & u_nv_2 & \cdots & u_nv_m \end{bmatrix}$$

- Matrix-Vector Multiplication

  v = **Au (A** is of size *n* x *m*, **u** is an *m*-element, **v** is an *n*-element vector)

$$v_i = \sum_{j=1}^{m} a_{ij}u_j$$

2 Matrix Algorithms - continued

- Matrix Multiplication

C = A B (**A** is $m \times n$, **B** is $n \times p$, **C** becomes $m \times p$)

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

- Solving Linear Systems

n lin. equations, n unknowns. Find $n \times 1$ vector **x**:

$$\mathbf{A}\,\mathbf{x} = \mathbf{y}$$

$$\mathbf{x} = \mathbf{A}^{-1}\,\mathbf{y}$$

number of computations for $\mathbf{A}^{-1}$ is high, procedure unstable.

Triangularize **A** to get upper triangular matrix $\mathbf{A}'$

$$\mathbf{A}'\,\mathbf{x} = \mathbf{y}'$$

back substitution provides solution **x**

2 Matrix Algorithms - continued

Matrix triangularization

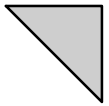
- Gaussian elimination
- LU decomposition
- QR decomposition

QR decomposition: orthogonal transform, e.g. Given's rotation (GR)

A = Q R

upper triangular M

M with orthonormal columns

A sequence of GR plane rotations annihilates **A**'s subdiagonal elements, and invertible **A** becomes an  matrix, **R**.

2 Matrix Algorithms - continued

$$\mathbf{Q}^{\mathbf{T}}\,\mathbf{A} = \mathbf{R}$$

$$\mathbf{Q}^{\mathbf{T}} = \mathbf{Q}_{(N-1)}\,\mathbf{Q}_{(N-2)} \cdots \mathbf{Q}_{(1)}$$

and

$$\mathbf{Q}_{(p)} = \mathbf{Q}^{(p,p)}\,\mathbf{Q}^{(p+1,p)} \ldots \mathbf{Q}^{(N-1,p)}$$

where $\mathbf{Q}^{(pq)}$ is the GR operator to nullify matrix element at the

(q+1)st row, pth column,

and has the following form:

2 Matrix Algorithms - continued

$$\mathbf{Q}^{(q,p)} = \begin{bmatrix} 1 & 0 & \cdots & & & 0 & 0 \\ 0 & 1 & \cdots & & & 0 & 0 \\ 0 & 0 & \cdots & & & 0 & 0 \\ \vdots & \vdots & & \cos\phi & \sin\phi & & \vdots \\ & & & -\sin\phi & \cos\phi & & \\ 0 & 0 & & & & & \vdots \\ 0 & 0 & \cdots & & & 1 & 0 \\ 0 & 0 & \cdots & & & 0 & 1 \end{bmatrix}$$

col. p ↓ col. p+1 ↓

← row q

← row q+1

where $\phi = \tan^{-1}\left[a_{q+1,p} / a_{q,p}\right]$

2 Matrix Algorithms - continued

$\mathbf{A'} = \mathbf{Q}^{(q,p)}$ then becomes:

$$a'_{q,k} = a_{q,k}\cos\phi + a_{q+1,k}\sin\phi$$

$$a'_{q+1,k} = -a_{q,k}\sin\phi + a_{q+1,k}\cos\phi$$

$$a'_{jk} = a_{jk} \quad \text{if} \quad j \neq q, q+1$$

for all $k = 1, \ldots, N$.

Back substitution

$$\mathbf{A'\,x} = \mathbf{y'}$$

$\mathbf{x}$ can be found heuristically. Example:

$$\begin{bmatrix} 1 & 1 & -1 \\ 0 & 3 & -2 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -2 \\ -9 \\ 3 \end{bmatrix}$$

Thus

$$\begin{array}{rrrcr} x_1 & +x_2 & -x_3 & = & -2 \\ & 3x_2 & -2x_3 & = & -9 \\ & & x_3 & = & 3 \end{array}$$

2 Matrix Algorithms - continued

- Iterative Methods

  When large, sparse matrices (e.g. $10^5 \times 10^5$) are involved

  $\mathbf{g} = \mathbf{H}\,\mathbf{f}$

  representing phys. measurements

Splitting: $\mathbf{A} = \mathbf{S} + \mathbf{T}$

initial guess: $\mathbf{x_0}$

iteration: $\mathbf{S}\,\mathbf{x_{k+1}} = -\mathbf{T}\mathbf{x_k} + \mathbf{y}$

Sequence of vectors $\mathbf{x_{k+1}}$ are expected to converge to $\mathbf{x}$.

2 Matrix Algorithms - continued

- Eigenvalue - Decomposition

A is of size *n* x *n*. If there exists **e** such that

$$\mathbf{A}\,\mathbf{e} = \lambda\,\mathbf{e}$$

λ is called eigenvalue, **e** is eigenvector.

λ obtained by solving the $|\mathbf{A} - \lambda\,\mathbf{I}| = 0$ characteristic eqn.

For distinct eigenvalues:

$$\mathbf{A}\,\mathbf{E} = \mathbf{E}\,\Lambda = \begin{bmatrix}\mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_n\end{bmatrix} \cdot \begin{bmatrix}\lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \lambda_n\end{bmatrix}$$

E is invertible, and hence $\mathbf{A} = \mathbf{E}\,\Lambda\,\mathbf{E}^{-1}$

n x *n* normal matrix **A**, i.e. $\mathbf{A}^{\mathbf{H}}\,\mathbf{A} = \mathbf{A}\,\mathbf{A}^{\mathbf{H}}$ can be factored

$$\mathbf{A} = \mathbf{U}\,\Lambda\,\mathbf{U}^{\mathbf{T}}$$

U is *n* x *n* unitary matrix. Spectral decomposition, KL transform.

2 Matrix Algorithms - continued

- Singular Value Decomposition (SVD)
  Useful in
  —image coding
  —image enhancement
  —image reconstruction, restoration - based on the pseudoinverse

$$\mathbf{A} = \mathbf{Q_1}\,\Sigma\,\mathbf{Q_2}^{T}$$

where $\mathbf{Q_1}$: $m \times m$ unitary M
$\mathbf{Q_2}$: $n \times n$ unitary M

$$\Sigma = \begin{bmatrix} \mathbf{D} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$$

where $\mathbf{D} = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r)$, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r, > 0$,
r is the rank of $\mathbf{A}$

2 Matrix Algorithms - continued

SVD can be rewritten:

$$\mathbf{A} = \mathbf{Q_1}\, \Sigma\, \mathbf{Q_2}^{\mathbf{T}} = \Sigma\,_{i=1}^{r} \sigma_i\, u_i\, v_i^T$$

where u_i is column vector of $\mathbf{Q_1}$,
v_i is column vector of $\mathbf{Q_2}$

The singular values of $\mathbf{A}$: $\sigma_1, \sigma_2, \ldots, \sigma_r$ are
square roots of the eigenvalues of $\mathbf{A^T A}$ (or $\mathbf{A A^T}$)

The column vectors of $\mathbf{Q_1}$, $\mathbf{Q_2}$ are the singular vectors of $\mathbf{A}$, and are eigenvectors of $\mathbf{A^T A}$ (or $\mathbf{A A^T}$).

SVD also used to

- solve the least squares problem
- determine the rank of a matrix
- find good low-rank approx. to the original matrix

2 Matrix Algorithms - continued

- Solving Least Squares Problems

  Useful in control, communication, DSP

  — equalization
  — spectral analysis
  — adaptive arrays
  — digital speech processing

  Problem formulation:

  Given **A** , an $n \times p$ ($n > p$, rank = p) observation matrix
  and **y**, an n-element desired data vector
  Find **w**, a p-element weight vector, which minimizes
  Euclidean norm of residual vector, **e**.

$$\mathbf{e} = \mathbf{y} - \mathbf{A}\,\mathbf{w}$$

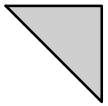
2 Matrix Algorithms - continued

Unconstrained Least Squares Algorithm

$$\mathbf{Q\,e = Q\,y - [Q\,A\,] = y' - A'\,w}$$

orthonormal M

upper triangular M

i.e. **A'** reduced to  represented by

$$\mathbf{A'} = \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix}$$

To minimize Euclidean norm of $\mathbf{y' - A'\,w}$, $\mathbf{w}_{opt}$ is obtained (**w** has no influence on the lower parts of the difference). Therefore

$$\mathbf{R}\ \mathbf{w}_{opt} = \mathbf{y'}$$

$\mathbf{w}_{opt}$ is obtained by back-substitution (**R** is)

3 Digital Signal Processing Algorithms

- Discrete Time Systems and the Z-transform

  Continuous - discrete time signals (sampled continuous signal)

  Linear Time Invariant (LTI) systems
  characterized by *h(n)*, the response to sampling sequence, $\delta(n)$.

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k) = x(n) * h(n)$$

  This is the *convolution* operation.

  Z-transform -- definition:

$$X(z) = Z[x(n)] = \sum_{n=-\infty}^{\infty} x(n)z^{-n}$$

  z is a complex number in a region of the z-plane.

3 DSP Algorithms - continued

Useful properties:

(i) $x(n) * h(n) \leftrightarrow X(z) \cdot H(z)$

(ii) $x(n + n_0) \leftrightarrow z^{n_0} \cdot X(z)$

- Convolution

$$y(n) = \sum_{k=-\infty}^{\infty} u(k)w(n-k) = u(n) * w(n)$$

computation:

$$y(n) = \sum_{k=0}^{N-1} u(k)w(n-k)$$

where $n = 0, 1, 2, \ldots, 2N\text{-}2$

u(n) . . . input sequence,
w(n). . . impulse response of digital filter
y(n) . . . processed (filtered) signal

3 DSP Algorithms - continued

Computation

Using transform (e.g. FFT) method, order of computation reduced

from $O(N^2)$ to $O(N \log N)$.

Recursive equations

$$y_j^k = y_j^{k-1} + u_k \, w_{j-k}$$

$k = 0, 1, ..., j$ when $j = 0, 1, ..., N\text{-}1$ and
$k = j - N + 1, \; j - N + 2, ..., N - 1$, when $j = N, N + 1, ..., 2N\text{-}2$

- Correlation $$y(n) = \sum_{k=-\infty}^{\infty} u(k) w(n+k)$$

computation:

$$y(n) = \sum_{k=0}^{N-1} u(k) w(n+k)$$

3 DSP Algorithms - continued

- Digital FIR and IIR Filters

$$H(e^{j\omega}) = |H(e^{j\omega})|\, e^{j\theta(\omega)}$$

$|H(e^{j\omega})|$ is the magnitude, $\theta(\omega)$ is the phase response.

— Finite Impulse Response (FIR)
— Infinite Impulse Response (IIR)

} filters

Representation: pth order difference eqn.

$$y(n) = \sum_{k=1}^{p} a_k y(n-k) + \sum_{k=0}^{q} b_k x(n-k)$$

$x(n)$ . . . input signal

$y(n)$ . . . output signal

— Moving Average Filter
— Autoregressive Filter
— Autoregressive Moving Average Filter

3 DSP Algorithms - continued

- Linear Phase Filter

$$\theta(\omega) = \alpha\,\omega$$

Impulse response of FIR filter:

$$h(n) = h(N - 1 - n), \quad n = 0, 1, \ldots, N - 1$$

Half number of multiplications can be used. For N odd:

$$H(z) = \sum_{n=0}^{N-1} h(n)z^{-n} = \sum_{n=0}^{(N-1)/2} h(n)z^{-n} + \sum_{n=(N+1)/2}^{N-1} h(n)z^{-1}$$

let $n' = (N-1) - n$

$$H(z) = \sum_{n=0}^{(N-1)/2} h(n)z^{-n} + \sum_{n'=0}^{(N-1)/2} h(n')z^{-(N-1-n')} =$$

$$= \sum_{n=0}^{(N-1)/2} h(n)\left[z^{-n} + z^{-(N-1-n)}\right]$$

3 DSP Algorithms - continued

- Discrete Fourier Transform (DFT)

  The DFT of finite lengths sequence *x(n)* is:

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{nk}$$

  where $k = 0, 1, 2, \ldots, N - 1$, and $W_N = e^{-j2\pi/N}$.

  Efficiently computed using FFT.

  Properties:

  — Obtained by uniformly sampling the FFT of the sequence at

$$\omega = 0, \frac{2\pi}{n}, 2 \cdot \frac{2\pi}{n}, \cdots, (N-1) \cdot \frac{2\pi}{n}$$

3 DSP Algorithms - continued

Inverse DFT:

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-nk}, \quad n = 0,1,\ldots, N-1$$

— Multiplying the DFTs of two *N*-point sequences is equivalent to the circular convolution of the two sequences:

$X_1(k)$ = DFT of $[x_1(n)]$

$X_2(k)$ = DFT of $[x_2(n)]$, then

$X_3(k) = X_1(k)\, X_2(k)$, is the DFT of $[x_3(n)]$

where

$$x_3(n) = \sum_{m=0}^{N-1} \tilde{x}_1(m) \tilde{x}_2(n-m)$$

and $n = 0, 1, ..., N-1$

3 DSP Algorithms - continued

- Fast Fourier Transform (FFT)

DFT computational complexity (direct method):

each $x(n)W^{nk}$ requires 1 complex multiplication

$X(k)$ $\{k = 0, 1, ..., N-1\}$ requires N^2 complex mult. + $N(N-1)$ addn.

DFT computational complexity using FFT ($N = 2^m$ case):

Utilizing symmetry + periodicity of W^{nk}, op. count reduced from N^2 to $N \log_2 N$

If one complex multiplication takes 0.5 μsec:

N	T_{DFT}	T_{FFT}
2^{12}	8 sec.	0.013 sec.
2^{16}	0.6 hours	0.26 sec.
2^{20}	6 days	5 sec.

3 DSP Algorithms - continued

— Decimation in time (DIT) algorithm /discussed here/
— Decimation in frequency (DIF)

DIT FFT

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} = \sum_{n=even}^{N-1} x(n) W_N^{nk} + \sum_{n=odd}^{N-1} x(n) W_N^{nk}$$

substituting $n = 2r$ for even, $n = 2r + 1$ for odd n,

$$X(k) = \sum_{r=0}^{N/2-1} x(2r) W_N^{2rk} + \sum_{r=0}^{N/2-1} x(2r+1) W_N^{(2r+1)k}$$

$$= \sum_{r=0}^{N/2} (W_N^2)^{rk} + W_N^k \sum_{r=0}^{N/2-1} x(2r+1)(W_N^2)^{rk}$$

3 DSP Algorithms - continued

since

$$W_N^2 = e^{-2j(2\pi/N)} = e^{-2j\pi/(N/2)} = W_{N/2}$$

$$X(k) = \sum_{r=0}^{N/2-1} x(2r) W_{N/2}^{rk} + W_N^k \sum_{r=0}^{N/2-1} x(2r+1) W_{N/2}^{rk} =$$

$$= G(k) + W_N^k H(k)$$

obtained via $N/2$-point FFT

- N-point FFT: via combining two $N/2$-point FFTs
- Applying this decomposition recursively, 2-point FFTs could be used.

FFT computation consists of a sequence of "***butterfly***"operations,
each consisting 1addition, 1 subtraction and 1 multiplication.

3 DSP Algorithms - continued

Linear convolution using FFT

(1) Append zeros to the two sequences of lengths N and M, to make them of lengths an integer power of two that is larger than or equal to $M+N-1$.

(2) Apply FFT to both zero appended sequences

(3) Multiply the two transformed domain sequences

(4) Apply inverse FFT to the new multiplied sequence

3 DSP Algorithms - continued

- Discrete Walsh-Hadamard Transform (WHT)

  Hadamard matrix: a square array of +1's and -1's, an orthogonal M.

  iterative definition:

$$\mathbf{H_2} = \frac{1}{\sqrt{2}} \cdot \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad \text{and} \quad \mathbf{H_{2N}} = \frac{1}{\sqrt{2}} \cdot \begin{bmatrix} \mathbf{H_N} & \mathbf{H_N} \\ \mathbf{H_N} & -\mathbf{H_N} \end{bmatrix}$$

  Size eight Hadamard matrix:

$$\mathbf{H_8} = \frac{1}{2\sqrt{2}} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

  Input data vector $\mathbf{x}$ of lengths N ($N=2^n$). Output $\mathbf{y} = \mathbf{H}_N \mathbf{x}$

4 Image Processing Algorithms

IP operations , which are extended forms of their 1D counterparts:

2D convolution:

$$y(n_1,n_2) = \sum_{k_1=0}^{n_1} \sum_{k_2=0}^{n_2} u(k_1,k_2)w(n_1-k_1,n_2-k_2)$$

where $n_1, n_2 \in \{ 0, 1, ..., 2N\text{-}2 \}$

2D correlation:

$$y(n_1,n_2) = \sum_{k_1=0}^{n_1} \sum_{k_2=0}^{n_2} u(k_1,k_2)w(n_1+k_1,n_2+k_2)$$

where $n_1, n_2 \in \{ -N+1, -N+2, ..., -1,0,1, ..., 2N\text{-}2 \}$

No of computations high -> transform methods are used

4 Image Processing Algorithms - continued

Two-dimensional filtering

Represented by

- 2D difference eqn. (space domain)
- transfer function (freq. domain)

Computation

- Fast 2D convolution, via 2D FFT
- 2D difference eqn. directly

Occasionally, successive 1D filtering

4 Image Processing Algorithms - continued

2D DFT, FFT, and Hadamard Transform

2D DFT - similar to 1D case:

$$X(k_1, k_2) = \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1, n_2) \cdot W_N^{n_1 k_1 + n_2 k_2}$$

where $k_1, k_2 \in \{0,1,2,\ldots,N-1\}$ *and* $W_N = e^{-j2\pi/N}$

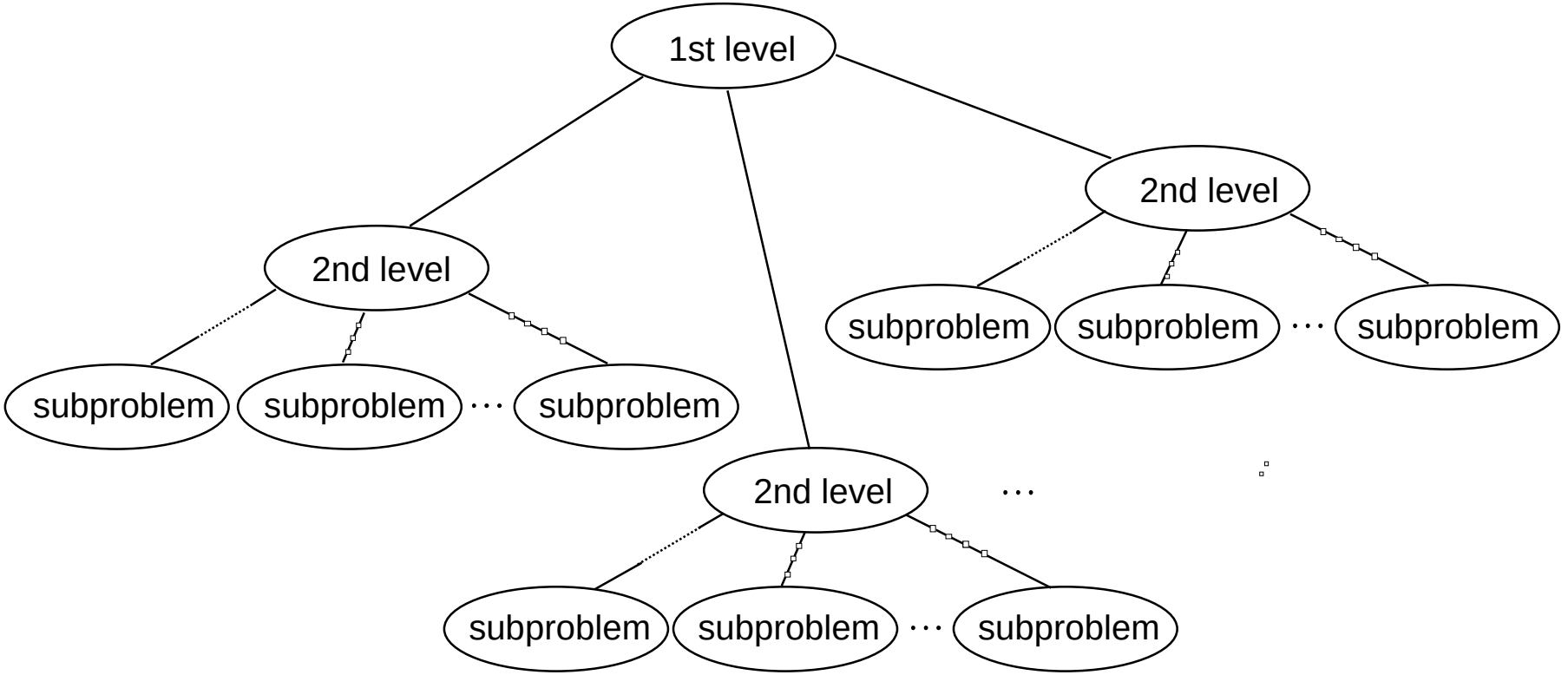
2D DFT can be calculated by

- applying N-times 1D FFT and N-times 1D FFT on the transformed sequence (= 2D FFT)
- transform methods: 2D FFT+ multiplication + 2D inverse FFT

2D Hadamard transform defined similarly

5 Advanced Algorithms and Applications

- Divide-and-Conquer Technique



5 Advanced Algorithms - continued

- Subproblems formulated like smaller versions of original
- same routine used repeatedly at different levels
- *top down*, *recursive* approach

Examples:

- sorting
- FFT algorithm

Important research topic

- design of interconnection networks
  (See FFT in “VLSI Array Algorithms” later)

5 Advanced Algorithms - continued

- Dynamic Programming Method
  - — Used in *optimization* problems to minimize/maximize a function
  - — *Bottom up* procedure
    Results of a stage used to solve the problems of the stage above
  - — One stage - one subproblem to solve
  - — Solutions to subproblems linked by *recurrence relation*
    important in *mapping* algorithms to arrays with *local interconnect*

  Examples:

  - — Shortest path problem in a graph
  - — Minimum cost path finding
  - — Dynamic Time Warping (for speech processing)

5 Advanced Algorithms - continued

- Relaxation Technique
  - — *Iterative approach*, making updating in parallel
  - — Each iteration uses data from most recent updating
    (in most cases neighboring data elements)
  - — Initial choices successively refined
  - — Very *suitable for array processors*, because it is order independent
    Updating at each data point *executed in parallel*

  Examples:

  - — Image reconstruction
  - — Restoration from blurring
  - — Partial differential equations

5 Advanced Algorithms - continued

- Stochastic Relaxation (Simulated Annealing)

  Problem in optimization approaches:
  solution may only be *locally* and not *globally optimal*

  — Energy function, state transition probability function introduced
  — Facilitates getting out of the trap of local optimum
  — Introduces trap flattening - based on *stochastic decision* temporarily accepting *worse* solutions
  — Probability of moving out of global optimum is low

  Examples:

  — Image restoration and reconstruction
  — Optimization
  — Code design for communication systems
  — Artificial intelligence

5 Advanced Algorithms - continued

- Associative Retrieval

  Features:

  — Recognition from partial information
  — Remarkable error correction capabilities
  — Based on Content Addressable Memory (CAM)
  — Performs *parallel search* and *parallel comparison* operations
  — Closely related to human brain functions

  Examples:

  — Storage, retrieval of rapidly changing database
  — image processing
  — computer vision
  — radar signal tracking
  — artificial intelligence

5 Advanced Algorithms - continued

Hopfield Networks

- Uses two-state ‘neurons’. In state *i*, outputs are: V_i^0 , V_i^1 .
- Inputs from a) external source I_i, and b) from other neurons
- Energy function given by:

$$E = -\sum_{i<j} T_{ij} V_i V_j - \sum_i I_i V_i$$

where T_{ij}: interconnection strength from neuron *i* to *j*

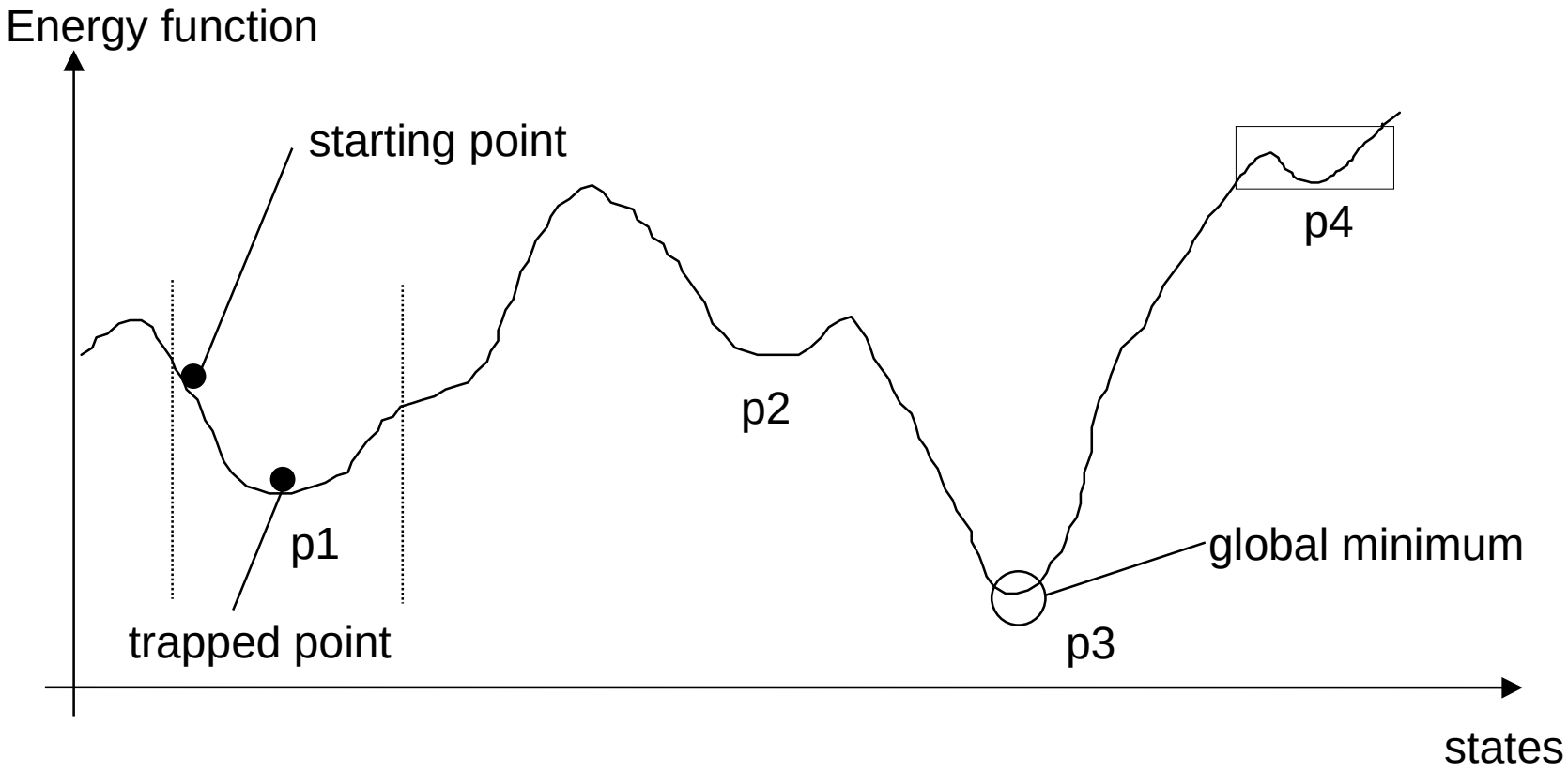
- Difference energy function between two different levels:

$$E_{i-on} - E_{i-off} = \Delta E_i = -\sum_j T_{ij} V_j - I_i$$

for $\Delta E < 0$, the unit turns on, for $\Delta E > 0$, the unit turns off

- The Hopfield model behaves as a CAM
  - Local minimum corresponds to stored *target pattern*.
  - Starting close to a stable state, it would converge to that state

5 Advanced Algorithms - continued



The original Hopfield model behaves as an associative memory. The local minimum (p1, p2, p3, p4) correspond to stored target patterns

6 VLSI Array Algorithms

Array algorithm: A set of rules for solving a problem in a finite number of steps by a multiple number of *interconnected* processors

— *Concurrency* achieved by decomposing the problem

- into independent subtasks executable in parallel
- into dependent subtasks executable in pipelined fashion

— *Communication* - most crucial regarding efficiency

- a scheme of moving data among PEs
- VLSI technology constrains *recursive* and *locally dependent* algorithms

— *Algorithm design*

- Understanding problem specification
- Mathematical / algorithmic analysis
- Dependence graph - effective tool
- New algorithmic design methodologies to exploit potential concurrency

6 VLSI Array Algorithms - continued

- Algorithm Design Criteria for VLSI Array Processors

  The effectiveness of mapping algorithm onto an array heavily depends on the way the algorithm is decomposed.

  - On *sequential machines* complexity depends on computation count and storage requirement
  - In *array proc. environment* overhead is non uniform, computation count is no longer an effective measure of performance

  — Area-Time Complexity Theory

  Complexity depends on computation time (T) and chip area (A)

  Complexity measure is AT^2 - not emphasized here, not recognized as a good design criteria

  Cost effectiveness measure $f(A,T)$ can be tailored to special needs

6 VLSI Array Algorithms - continued

—Design Criteria for VLSI Array Algorithms

New criteria needed to determine algorithm efficiency to include

- stringent *communication problems* associated with VLSI technology
- *communication costs*
- parallelism and *pipelining rate*

Criteria should comprise computation, communication, memory and I/O

Their key aspects are:

- **Maximum parallelism**
  which is exploitable by the computing array
- **Maximum pipelineability**
  For regular and locally connected networks
  Unpredictable data dependency may jeopardize efficiency
  Iterative methods, dynamic, data-dependent branching are less well suited to pipelined architectures

6 VLSI Array Algorithms - continued

- **Balance among computations, communications and memory**
  Critical to the effectiveness of array computing
  Pipelining is suitable for balancing computations and I/O
- **Trade-off between computation and communication**
  Key issues
    - local / global
    - static / dynamic
    - data dependent / data independent

  Trade-off between interconnection cost and thruput is to be maximized
- **Numerical performance, quantization effects**
  Numerical behavior depends on word lengths and algorithm
  Additional computation may be necessary to improve precision
  Heavily ‘problem dependent’ issue - no general rule

6 VLSI Array Algorithms - continued

- Locally and Globally Recursive Algorithms

  Common features of signal / image processing algorithms:

  - intensive computation
  - matrix operations
  - localized or perfect shuffle operations

  In an interconnected network each PE should know *when*, *where* and *how* to send / fetch data.

  where? In locally recursive algorithms data movements are confined to nearest neighbor PEs. Here locally interconnected network is OK

  when? In globally synchronous schemes timing controlled by a sequence of ‘beats’ (see systolic array)

6 VLSI Array Algorithms - continued

—Local and Global Communications in Algorithms

Concurrent processing performance critically depends on *communication cost*

Each PE is assigned a *location index*

Communication cost characterized by the *distance* between PEs

Time index, *spatial index* - to show when and where computation takes place

- **Local type** recursive algorithm: index separations are within a certain limit (E.g. matrix multiplication, convolution)
- **Global type** recursive algorithm: recursion involves separated space indices. Calls for globally interconnected structures
  (E.g. FFT and sorting)

6 VLSI Array Algorithms - continued

— Locally Recursive Algorithms

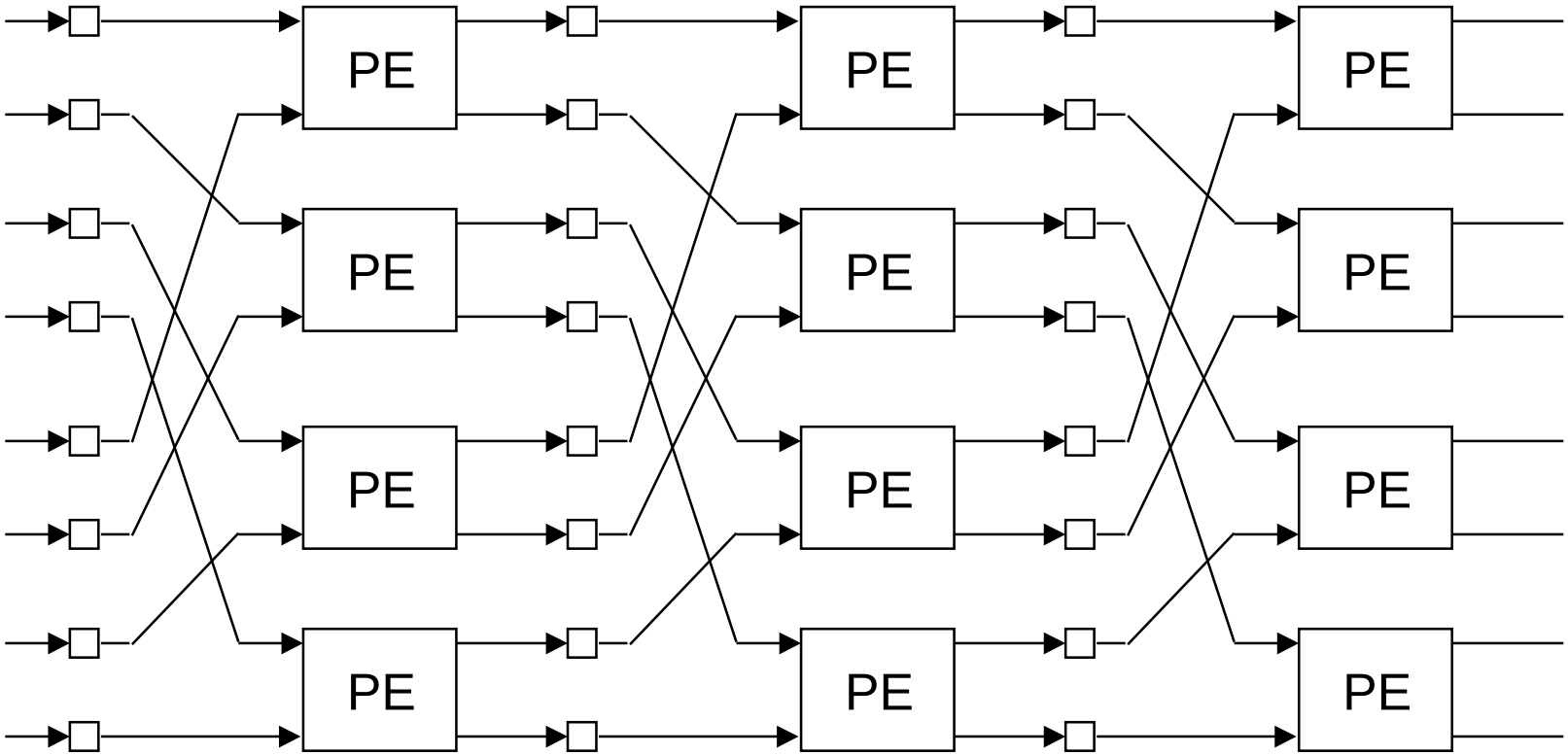
- Majority of algorithms: localized operations, intensive computation
- When mapped onto array structure only *local communication* required
- Next subject (chapter) will be entirely devoted to these algorithms

— Globally Recursive Algorithms: FFT example

- Perfect shuffle in FFT requires global communication
- $(N/2)\log_2 N$ butterfly operations needed
- For each butterfly

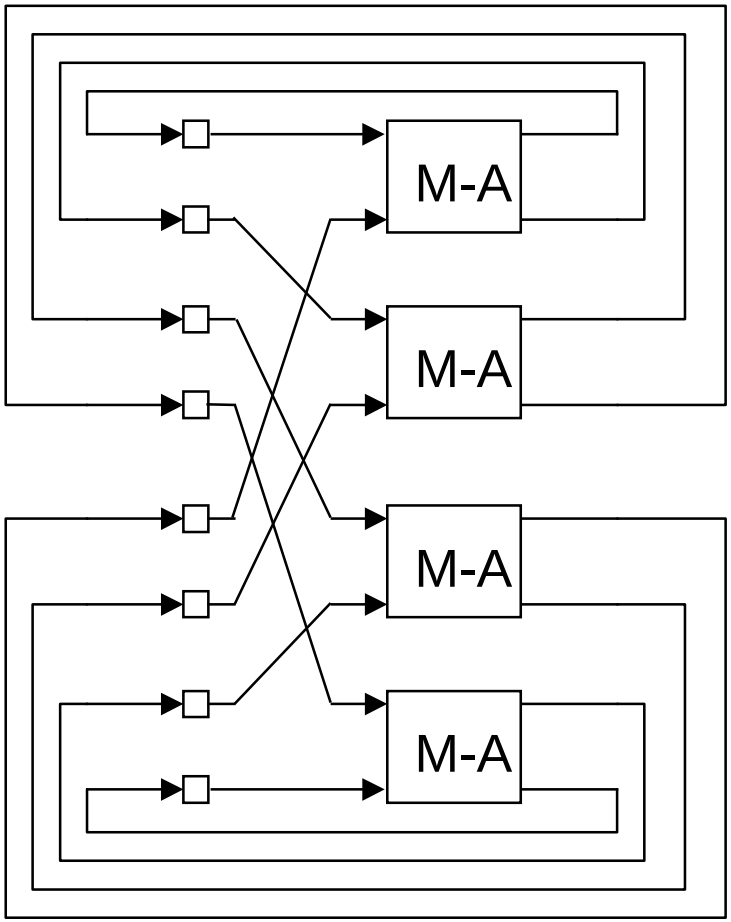
  4 real multiplications and
  4 real additions needed
- In single state configuration $N/2$ PEs and $\log_2 N$ time units needed

6 VLSI Array Algorithms - continued



Array configuration for the FFT computation:
Multistage Array

6 VLSI Array Algorithms - continued



Array configuration for the FFT computation:
Single-stage Array

6 VLSI Array Algorithms - continued

Perfect Shuffle Permutation

Single bit left shift of the binary representation of index *x:*

$$x = \{ b_n,, b_{n-1}, ..., b_1 \}$$

$$\sigma(x) = \{b_{n-1}, b_{n-2},, ..., b_1, b_n \}$$

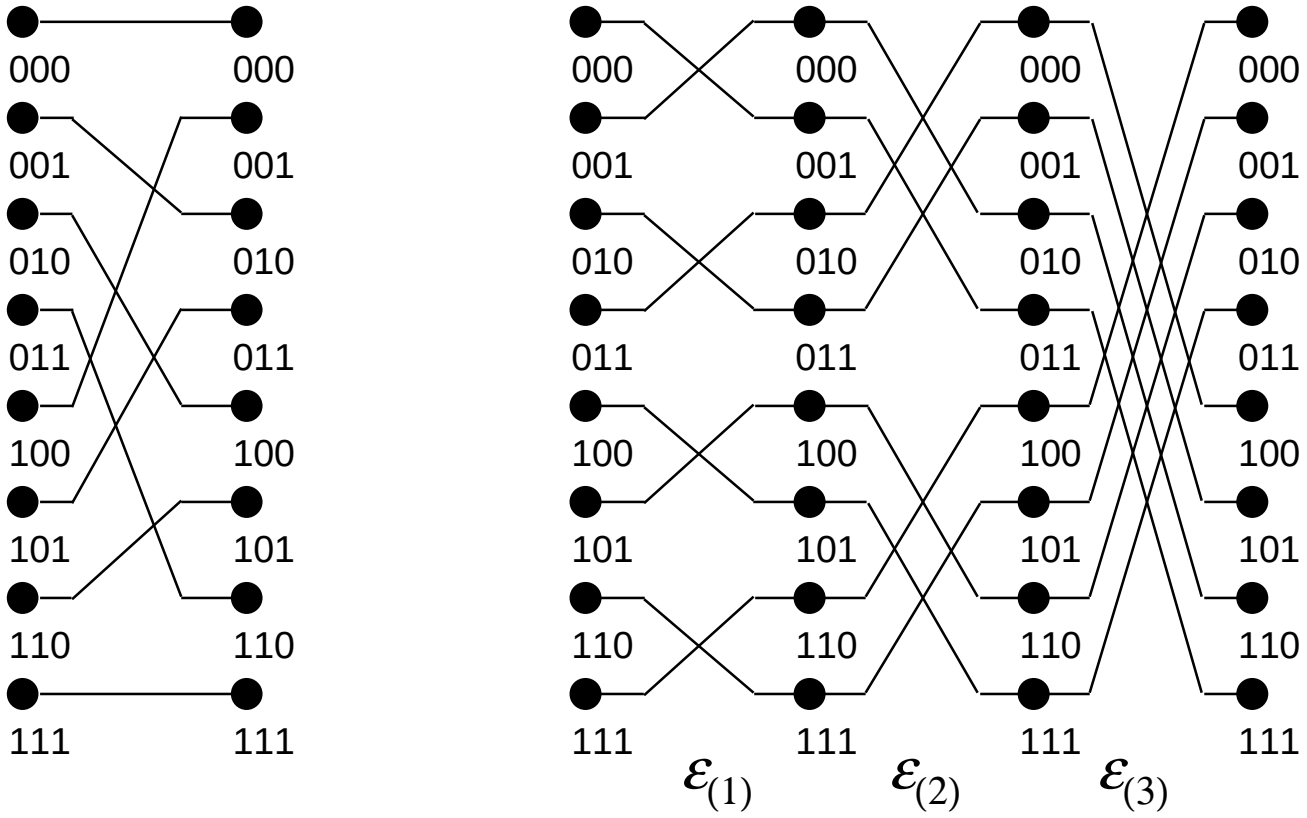
Exchange permutation

$$\varepsilon_{(k)}(x) = \{ b_n,, ..., b_k', ..., b_1 \}$$

where b_k' denotes the complement of the kth bit

The next figure compares perfect shuffle permutation and exchange permutation networks

6 VLSI Array Algorithms - continued



(a) (b)

(a) Perfect shuffle permutations, (b) Exchange permutations

6 VLSI Array Algorithms - continued

FFT via Shuffle-Exchange Network

Interconnection network for in-place computation has to provide

- exchange permutation ($\varepsilon_{(k)}$)
- bit-reversal permutation (ρ)

For a 8-point DIT FFT the interconnection network can be represented as

$$\rho[\varepsilon_{(1)}[\varepsilon_{(2)}[\varepsilon_{(3)}]]]$$

apply $\varepsilon_{(3)}$ first, $\varepsilon_{(2)}$ next, etc.

$X(k)$ computed by separating $x(k)$ into even and odd $N/2$ sequences

$$X(k) = \sum_{n=0}^{7} x(n) W_N^{nk}$$

n and k are represented by 3-bit binary numbers:

$$n = (n_3\, n_2 n_1) = 4n_3 + 2n_2 + n_1$$

$$k = (k_3\, k_2\, k_1) = 4k_3 + 2k_2 + k_1$$

6 VLSI Array Algorithms - continued

Result:

Original Index		Bit-reversed Index	
$x(0)$	000	$x(0)$	000
$x(1)$	001	$x(4)$	100
$x(2)$	010	$x(2)$	010
$x(3)$	011	$x(6)$	110
$x(4)$	100	$x(1)$	001
$x(5)$	101	$x(5)$	101
$x(6)$	110	$x(3)$	011
$x(7)$	111	$x(7)$	111

Due to in-place replacement (i.e. input and output data share storage)

n_1 is replaced by k_3,

n_3 is replaced by k_1, etc.

$x(n_3 n_2 n_1)$ is stored in the array position $X(k_1 k_2 k_3)$

i.e. to determine the position of $x(n_3 n_2 n_1)$ in the input, bits of index n have to be reversed.