Parallel Programming as an Optimization Problem or Intelligent Compilers for Parallel Computers

Geoffrey C. Fox
Vansanth Balasundaram

CRPC-TR90161
April 1990

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

Invited paper at Second International Conference on Expert Systems for Numerical Computing, April 25, 1990, Purdue University.

SCCS 110

C³P-891

April 27, 1990

Parallel Programming as an Optimization Problem or Intelligent Compilers for Parallel Computers

Geoffrey C. Fox and Vasanth Balasundaram

Caltech Concurrent Computation Program
California Institute of Technology, 206-49
Pasadena, California 91125

Second International Conference on Expert Systems
for Numerical Computing

April 25, 1990
Purdue University

In collaboration with Ken Kennedy and Uli Kremer
Rice University

Parallel Programming as an Optimization Problem

or Intelligent Compilers for Parallel Computers

Geoffrey Fox
Vasanth Balasundaram
Caltech → Syracuse (June - July)

April 25, 1990

Purdue Conference

Collaboration with Rice
Ken Kennedy, Uli Kremer

INTELLIGENT FORTRAN COMPILER FOR PARALLEL COMPUTERS

Vasanth Balasundaram and Geoffrey Fox

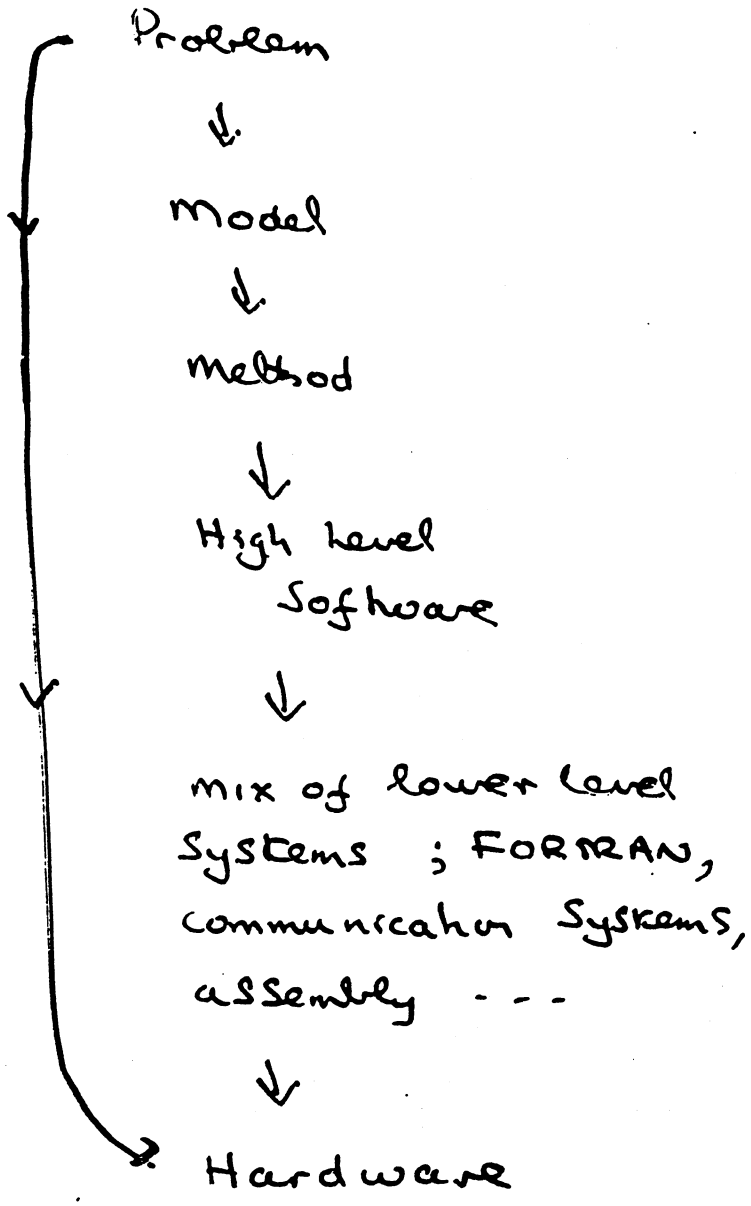
Abstract

For at least the next few years, FORTRAN will be a critical language for scientific computation. One can view it either as a primary user interface or as a portable "machine-language" for which excellent compilers exist and which is a target for more user-friendly systems. Mapping a scientific problem onto a high performance computer can be viewed as a hard optimization problem where one minimizes some combination of user program development time and production program execution time. We consider how expert systems, neural networks, simulated annealing, and related methods can be integrated into a FORTRAN compiler to both give better code and feedback to the user on the appropriateness of a particular problem to particular hardware.

- What is the problem and our approach?
- Some Approaches to Optimization
  - Combinatorial Optimization →
  - Neural Networks
- The "Expert System"
  - highest level ↑ which programs should use machines with what programming system?
- Structure of Compiler and Role of optimization
  - Statement level Scheduling
  - Typical block of loops
  - Overall Program

↓ increasing granularity

Scientific Computing



A hierarchy of mapping problems.

Would like to optimize the overall mapping

We will give a very preliminary discussion of where optimization is possible and what type of optimization methods could be used.

We believe powerful (new) optimization methods can be applied in several areas. This can lead to

Better Performance

New criteria for languages

New designs of software environments

. . . . . .

Many Approaches to Optimization

Heuristics — from the problem

Combinatorial Optimization — from mathematics

Expert Systems (classic AI) — from computer Science

Genetic algorithms — from nature

Annealing — from physics

Neural Networks — from biology

Elastic Networks

Deterministic Annealing

Information Theory (maximum Entropy) — from electrical engineering (astronomy?)

Physical Optimization

/

according to Webster, means

"pertaining to nature"

Is complexity analysis relevant to physical optimization methods?

We can also - and indeed this should be norm? - combine methods

e.g. nature

Genetic algorithms	evolve people over long time period
⋮	
expert system	high level reasoning
⋮	
learning networks and ?	middle level
⋮	
optimization networks	low level vision

Discrete Optimization by Physical Methods

Minimize $f(\eta)$

$\eta = \eta_0 \eta_1 \eta_2 \ldots$ is a set of binary (0-1) variables

Genetic Optimization Method

$\eta_1 = \alpha_0 \alpha_1 \ldots \alpha_n$

$\eta_2 = \beta_0 \beta_1 \ldots \beta_n$

Based on values $f(\eta_1)$, $f(\eta_2)$ produce child

$\alpha_0 \alpha_1 \ldots \alpha_m \beta_{m+1} \ldots \beta_n$

Natural for loosely coupled heterogeneous systems i.e. where different components of η label different parts (eye color, interest in physics..) of system.

Physics or Infomation Theory Approach

Probability of A is $\exp(-\beta f(q))$

$\beta = 1/T$, T temperature

as $\beta \to \infty$, state with minimum value for f has probability one.

Simulated Annealing

f labels states in a physical system

use standard monte carlo methods to obtain states at given β.

Lower T slowly (e.g. $T_k = .99 T_{k-1}$) and find ground state by annealing system.

At finite temperature, annealing gives the most likely state given that f is to be small

Neural Networks

$$\langle \eta \rangle = \frac{\sum_{\eta} \eta \, e^{-\beta f(\eta)}}{Z = \sum_{\eta} e^{-\beta f(\eta)}}$$

→ desired answer as $\beta \to 0$

At fixed $T \neq 0$, use "mean field" approximation to calculate right hand side. This gives deterministic equation

$$\langle \eta \rangle|_T = f(\langle \eta \rangle|_T)$$

Let $T \to 0$, this is neural network approach

- Physical optimization methods are usually easier to parallelize

Sometimes one has constraints

e.g. neural networks for scheduling

$\eta(a, l, t) = 1$ if variable a in location l at time t

$= 0$ otherwise

Then one minimizes

f + const. Penalty function if a in two locations at same time

This leads to poor results when many constraints e.g. Travelling Salesman Problem.

Kohler discussed in your last meeting

Best in cases when one CAN violate constraints / allow nonoptimality

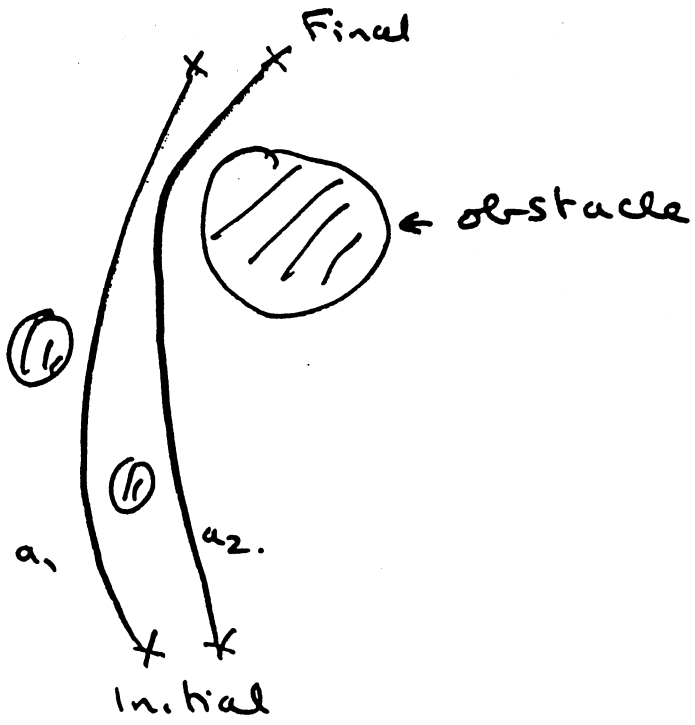
Simic showed how you could calculate mean field summing over just those η's that satisfy syntax constraints. This leads to equations involving new variables that satisfy constraints

$$\langle \underline{x}(a,t) \rangle = \tilde{\mathcal{G}} \left(\langle \underline{x}(a,t) \rangle, \beta \right)$$

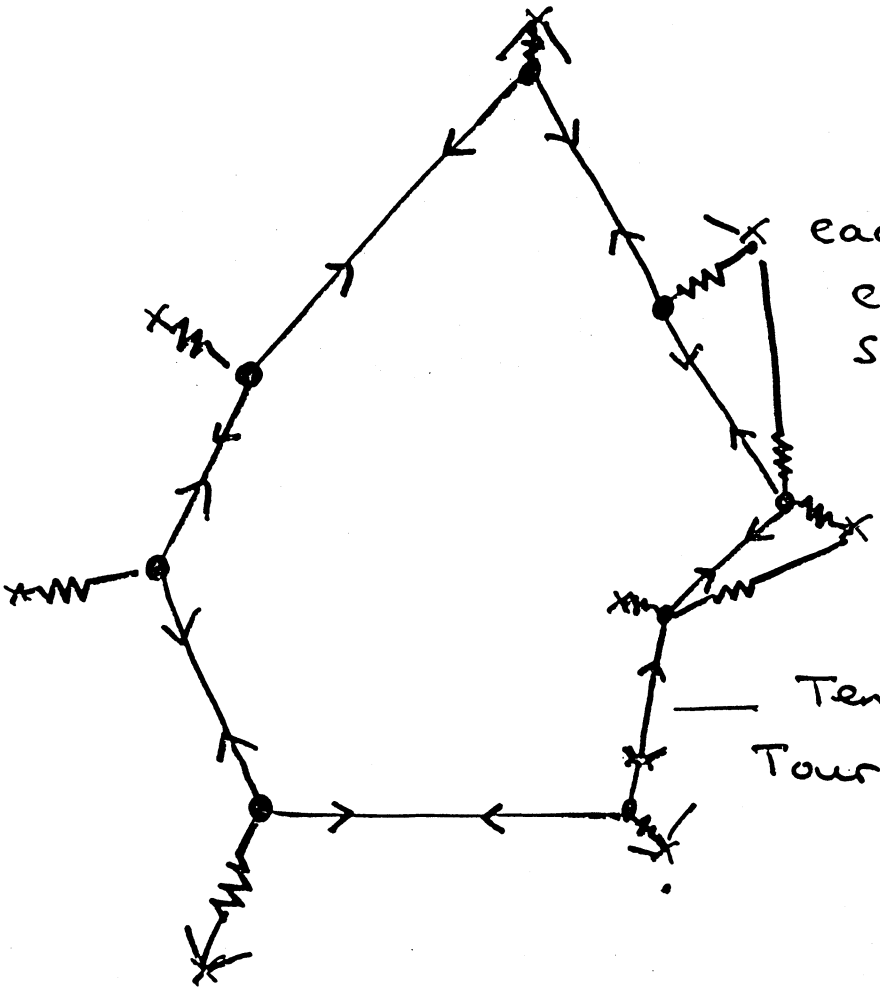
$\underline{x}(a,t)$ = location of a at time t

This is elastic net formalism which is a particle dynamics picture

e.g. Navigation



Travelling Salesman Problem



each city attracts each particle. Strongest force to nearest particle (these forces are constraints as ● ⇒ × eventually)

— Tension minimizes Tour length

● = Particles labelled by time

× = city

Progress in Solution of NP Complete Problems

TSP 90 Shoot-out

Blind testing of methods on a challenge 773 City Problem

Exact method: branch and Cut (Padberg...) Ophmal tour (with guarantee of ophmality) in 2½ hours

Physical Ophmizaten: Best implemetation is variant of Simulated Annealing + Lin-Kernighan heuristic

Ophmal tour in 6 hours (without guarantee). Very good Solution (within 2%) in minutes.

(Sparc Station was common machine)

Learning ("back propagation")

Neural Networks

- Robust method of "classifing"

Spaces

e.g. 64 pixels → Learning Net → classify as A → Z

- Need to train with samples
- Better than expert systems

for classifying numerically labelled data

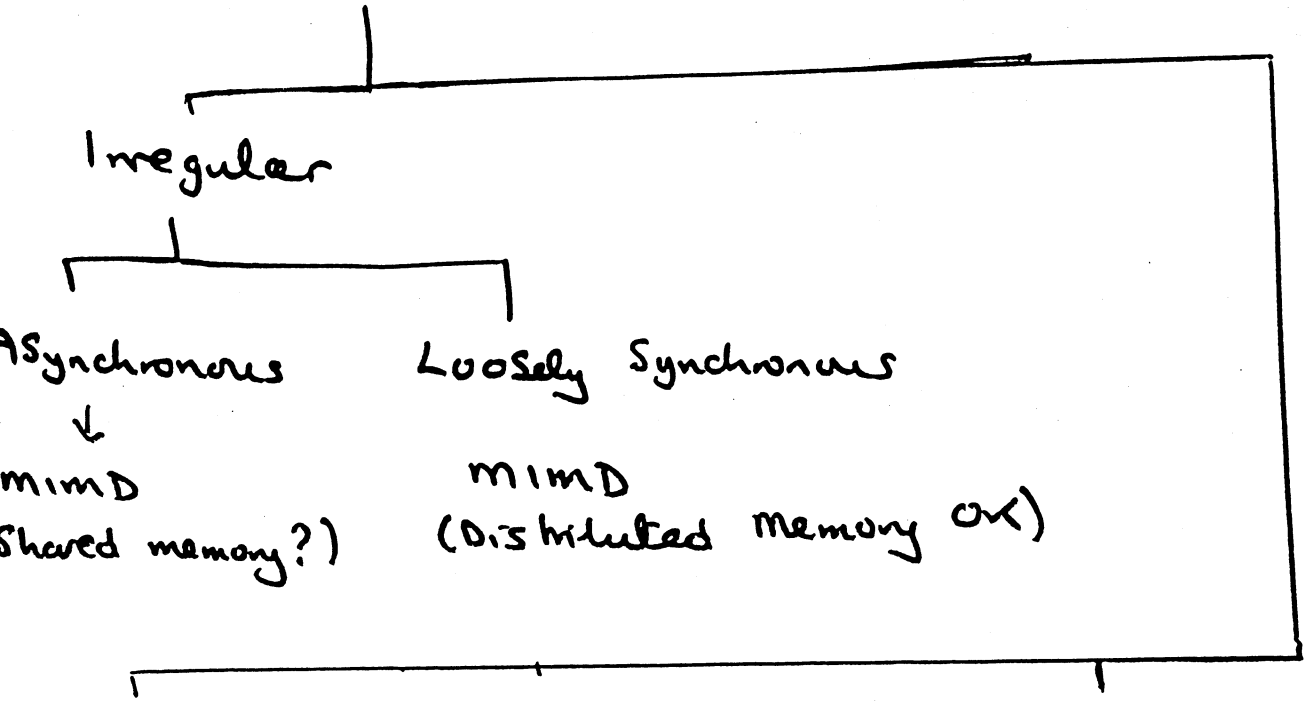
e.g. expected accuracy / speed of given algorithm of particular size and on a particular machine → Neural Net

whereas choice of algorithm → Expert System ?

The High Level (Expert System)

Summary of Caltech Work — My contribution to it!

Problem Structure



Regular
+ forall with branching (irregular)
↓
FORTRAN
↓
MIMD
Vector Supercomputer

Regular
↓
FORTRAN
(or FORTRAN 8X)
↓
SIMD
Vector Supercomputer
MIMD

Table 3.1 RELATIVE PERFORMANCE OF SELECTED CALTECH PARALLEL APPLICATIONS

			← MIMD →		← SIMD →		
Application	1 head CRAY XMP	8 head CRAY YMP	1024 node NCUBE-1 Hypercube	128 node Mark IIIfp Hypercube	4096 node DAP 610	64 K CM-2	Will Semi-Automatic Parallelizing (FORTRAN) Compiler Work?
QCD	1	12	1	6	x	50	Straightforward
Continuous Spin (High T_c)	1	12	1	6	x	20	Straightforward
Ising/Potts Models	1	12	32	10	3	30	Yes
Strings	1	12	8	6	x	x	Straightforward
Particle Dynamics							
$O(N \log N)$	1	12	4	16	x	x	doubtful
$O(N^2)$	1	12	1	6	x		Straightforward
Astronomical Data Analysis	IBM 3090 2 pulsars in 1989		NCUBE 5 pulsars in 1989	?	x	x	doubtful
Chemical Reactions							
$H + H_2$ Scattering	1	12	?	1.5	x	?	Possible
$e^- + CO$ Scattering	1	12	?	13	x	?	Possible
Grain Dynamics	1	12	1	6	x	15—30	Straightforward
Plasma Physics	1	12	?	2	x	?	Possible
Neural Networks	1	12	1	6	x	15—30	Possible
Computer Chess	1	12	2.5	?	x	x	no
Multi-target Tracking	1	12	x	12	x	x	hard

When can FORTRAN be parallelized by compiler

When FORTRAN data structures capture
i.e arrays
structure of problem = use data
parallelism

or
"embarassingly parallel" problem
e.g. calculation of matrix elements in
a chemical reaction
which can be captured by forall

Would like examples where one
can (with good compiler) parallelize
FORTRAN code and one CANNOT
naturally write algorithm in FORTRAN 8X
including forall.

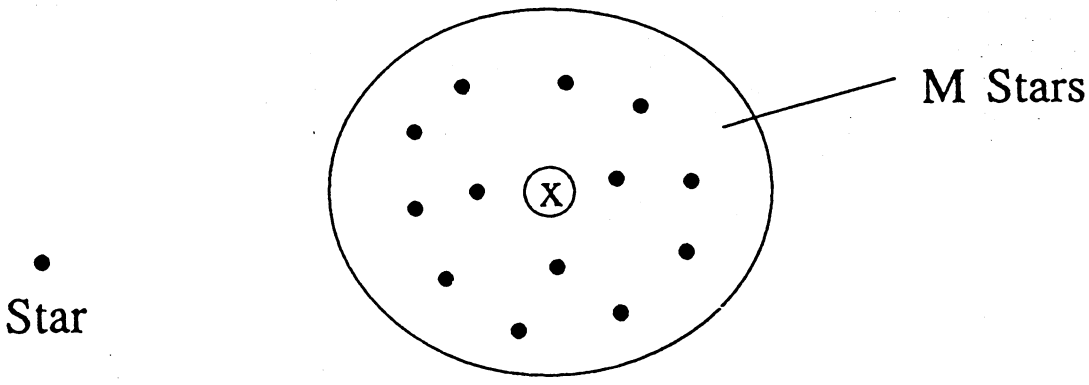
I don't know how to use parallelizing FORTRAN compiler on:
Irregular Problems which are either:

- Loosely Synchronous:
  Dynamic adaptive Finite Element
  Clustering algorithms for particle dynamics
  monte carlo
  (algorithms give irregular dynamic domains)
- ASynchronous
  Chess
  Event driven Simulations

This is my challenge to Programming languages researchers!
Currently we do parallelism by hand (or in ways that lose natural parallelism.)

Large N body Calculations

Quinn, Salmon, Warren, Space Telescope Institute, Caltech



Clustering algorithm (Appel, Barnes - Hut, Greengard)
Can replace M body force calculations by one using center of mass of cluster

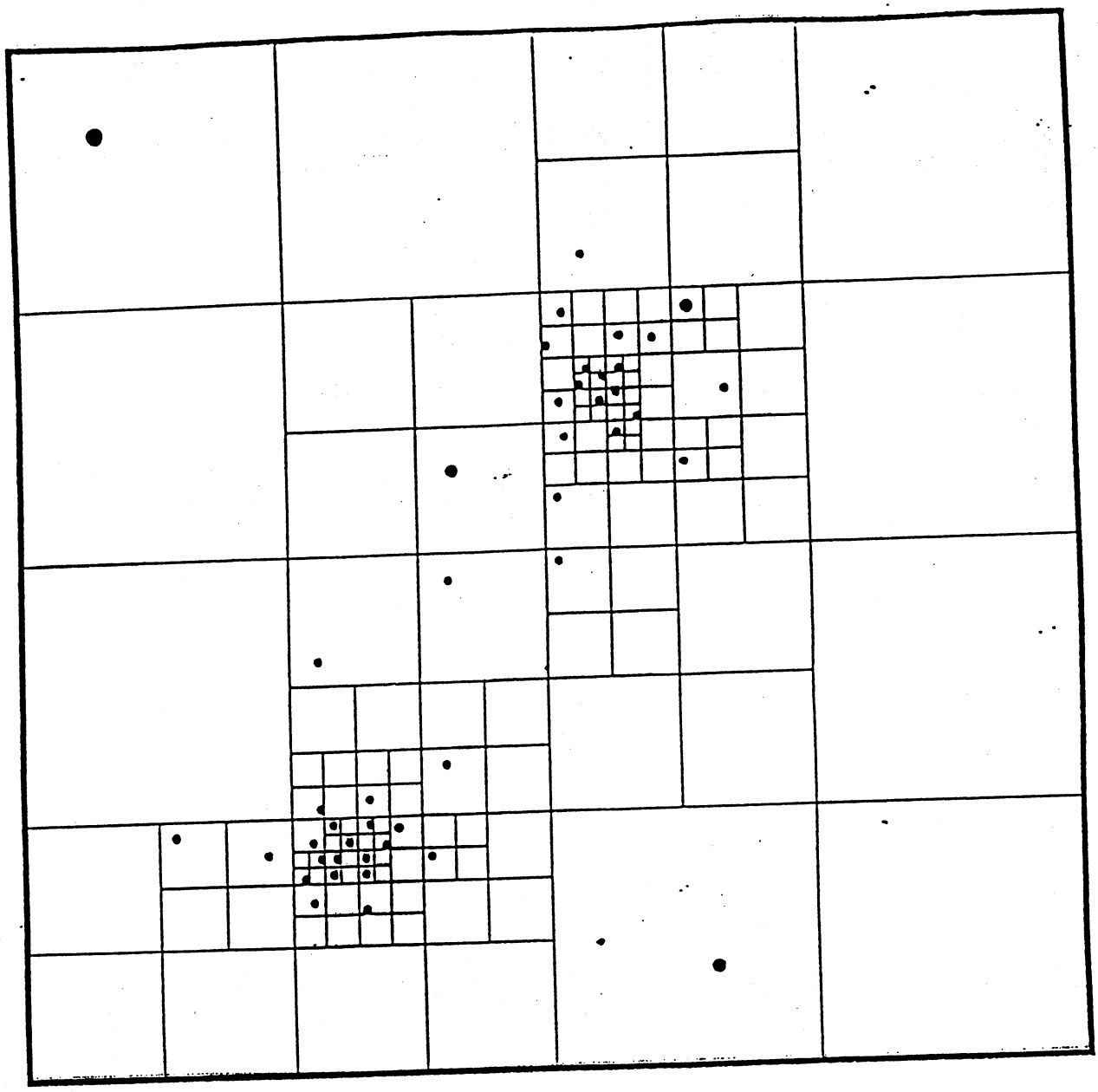
Naive calculation	$1/2\ N^2\ t_{2particle}$	
Clustering	$(20\text{-}50)\ N \log_2 N\ t_{2particle}$	best when $N \gtrsim 1000$

Can do O (10,000) Particles with $O(N^2)$ algorithm
One to two orders of magnitude larger problems possible with clustering algorithm

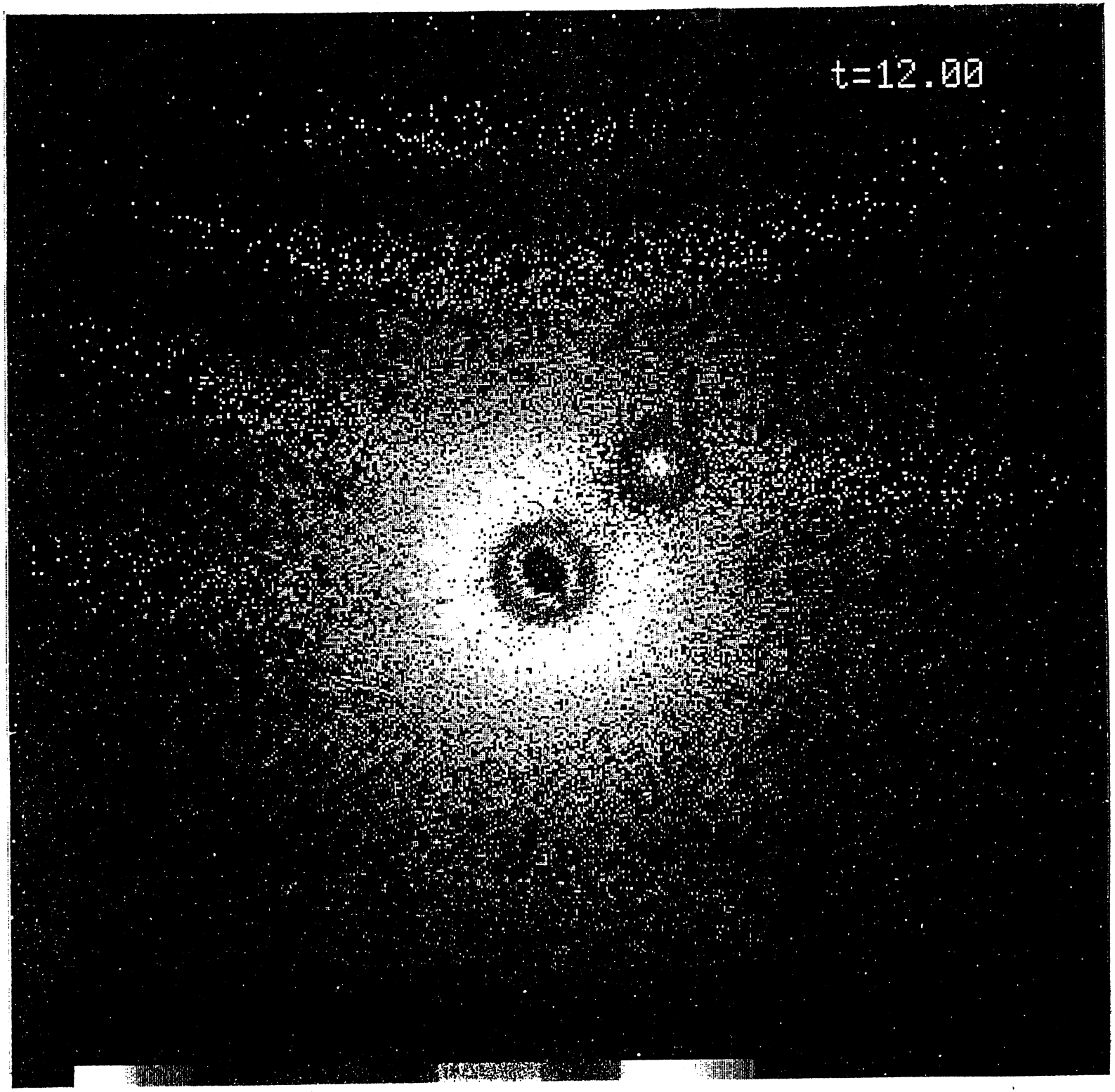
We had hoped to do $O(10^6)$ particles

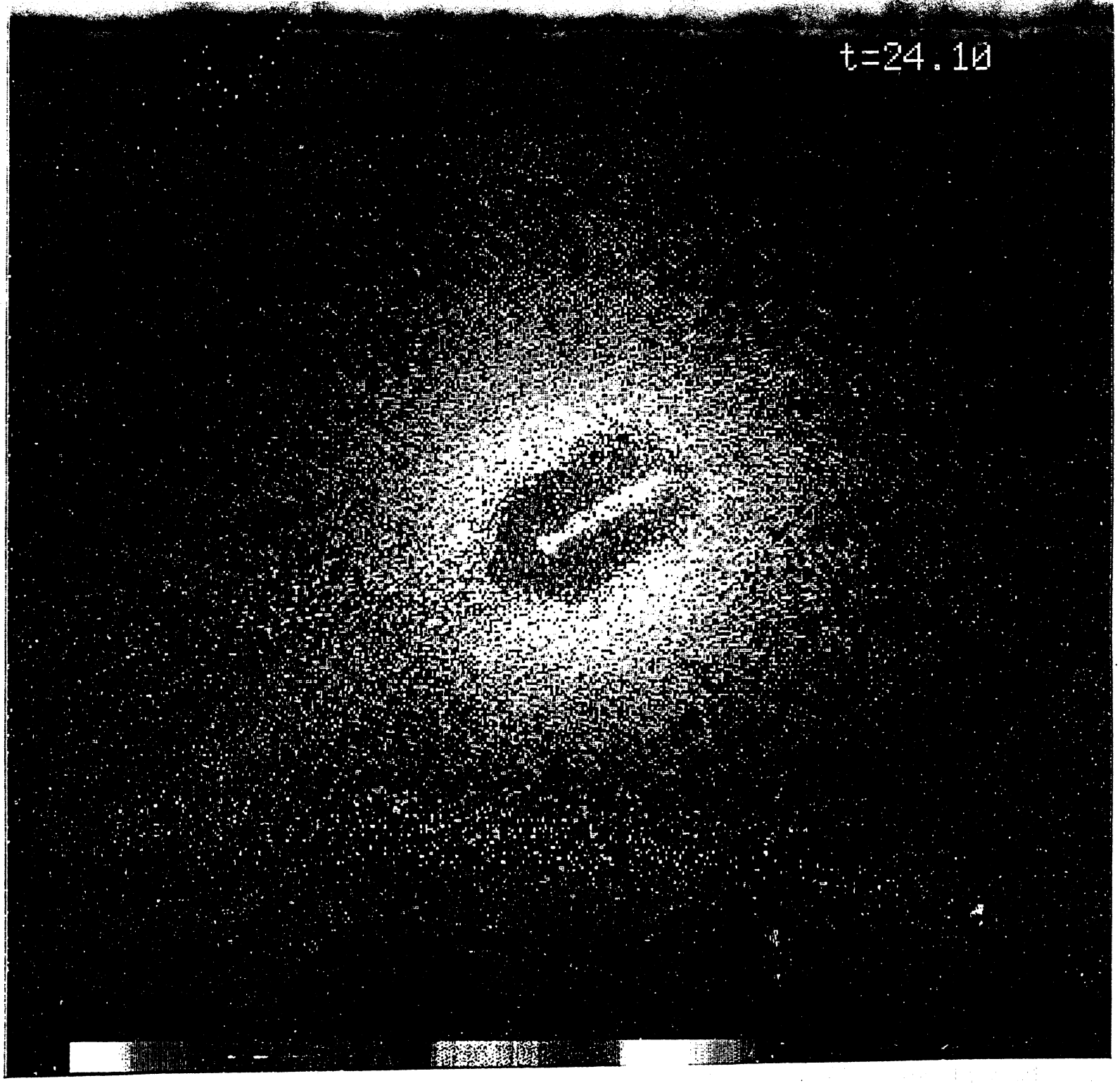
- Galaxy Structure and Collision of Galaxies
  Running 180,000 objects on 512 node NCUBE
- Cosmology - growth of fluctuations
- Globular Clusters - very difficult as short range interactions (binaries) important

Hierarchical Decomposition



Hierarchical boxing, presented for simplicity in two dimensions. A system of particles is shown, and the recursive subdivision of space induced by these particles.


t=12.00


t=24.10

OVERALL STRUCTURE OF SYSTEM

- Parallelizing FORTRAN compiler as a portable assembly language" for regular problems
- Pool of possible optimizations

  (a) optimizations that improve local processor performance.

  (b) optimizations that improve parallelism

  (c) optimizations that reduce communication overhead

  ⋮

- The objective function is an expression of the form $P_a + P_b + P_c + \ldots$

  where "P_x" is an estimate of the performance improvement when optimization "x" is performed.

- The estimate of "performance improvement" is done using a <u>performance model</u> that is based on the target machine's parameters.

- Goal is to <u>maximize</u> the objective function.

  ⇒ maximize performance improvement.

These optimizations may be done

- at compile-time by the compiler
- at compile-time by the user helping the compiler
- at run-time

- We will need to look at the program at different granularities.
- Data parallelism isolates a natural block size.

(A) ⇒ DO 1 I = 1, N1

DO 1 J = 1, N2

(B) ⇒ B(I, J) = (A(I-1, J) + A(I+1, J)
+ A(I, J+1) + A(I, J-1)) *.25

1 ENDDO

- Data Parallelism for loosely Synchronous (regular or irregular) problems ⇒ COLLECTIVE COMMUNICATION

i.e. at (A) not (B).

COLLECTIVE COMMUNICATION roles that message traffic is not random but highly correlated

- optimize mutual message traffic
- maximizes message sizes ⇒ reduces latency problems

(Almost) Optimal Deterministic Algorithms)

- COMBINE
- CONCAT
- EXCHANGE
- SHIFT
- :
- Personalized Communication
- :
- CRYSTAL ROUTER .

See "Solving Problems on Concurrent Processors" and most completely by Ho and Johnsson

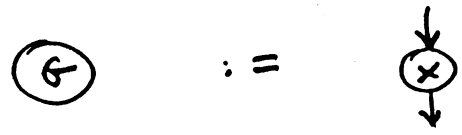
Dynamic Optimization

- NEURAL ROUTER

Supported by EXPRESS which is low level message passing system we will generate code for.

Bottom-up propagation of Data Window information:

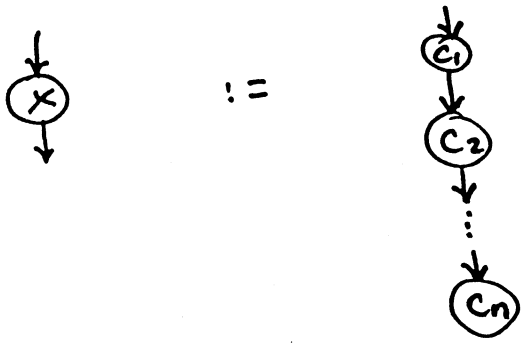
Graph grammars



label

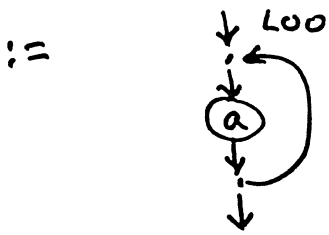
$L(G) = L(x)$

$\Delta(G) = \Delta(x)$ — data window



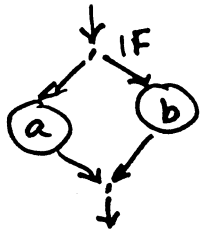
$L(x) = \prod_i L(C_i)$

$\Delta(x) = \bigcup_i \Delta(C_i)$



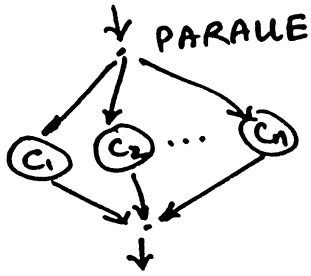
$L(x) = \overline{L(a)}$

$\Delta(x) = \Uparrow \Delta(a)$ — add loop inductor variable to $\Delta(a)$



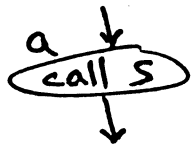
$L(x) = L(a) + L(b)$

$\Delta(x) = \Delta(a) \cup \Delta(b)$



$L(x) = L(C_1) \,||\, L(C_2) \,||\, \cdots \,||\, L(C_n)$

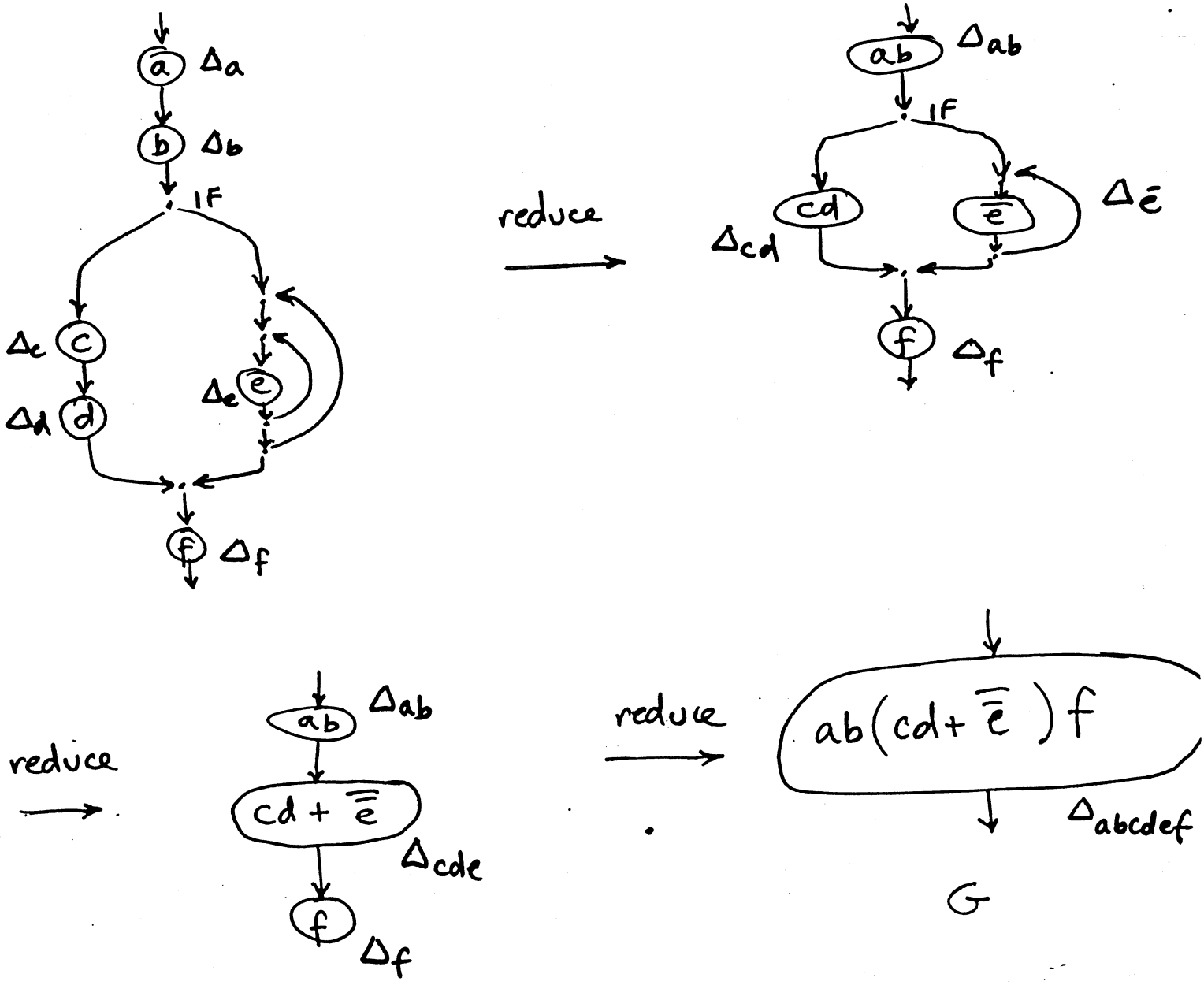
$\Delta(x) = \bigcup_i \Delta(C_i)$



$L(x) = \{L(a)\}$

$\Delta(x) = \Uparrow \Delta(S)$ — add parameter bindings to $\Delta(S)$

- Graph grammar for a language that allows only single-entry single-exit regions

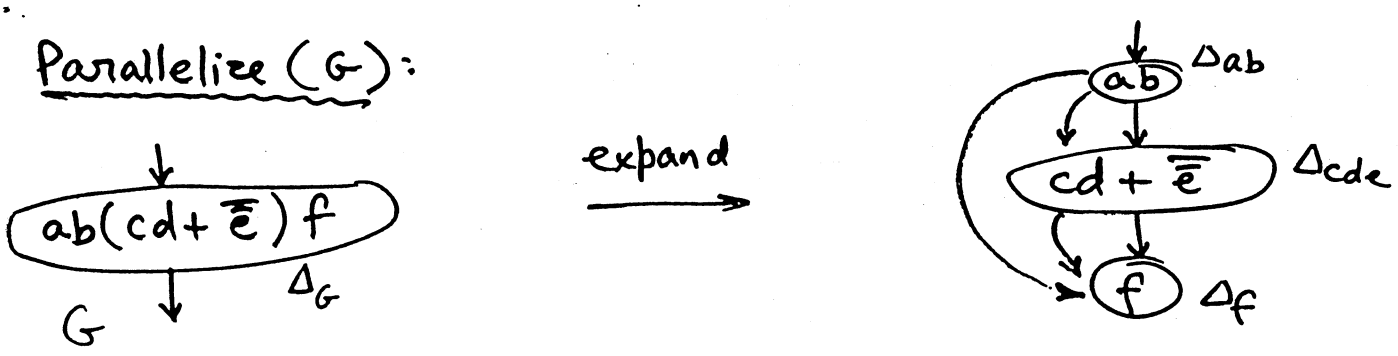


Note: Each reduction takes the program from a lower representation level to a higher representation level.

Also, Data Window information is "fed back" to the higher level from the lower level.

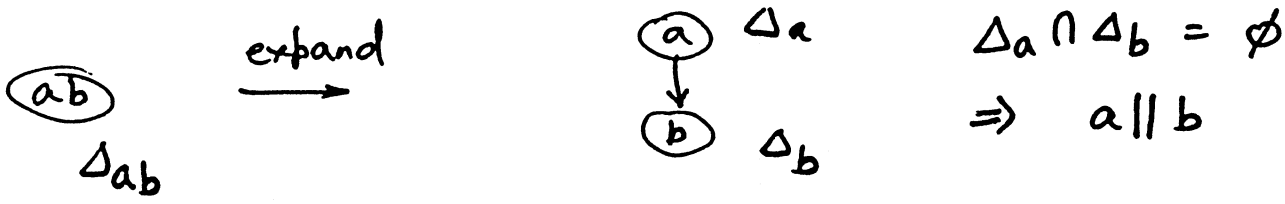
Top-down recursive parallelization

- **Parallelize (G):**



Level 0 Level 1

- **Parallelize (ab):**



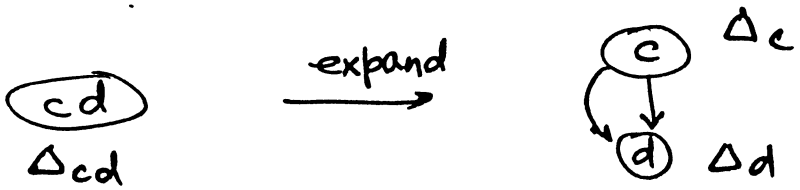
$\Delta_a \cap \Delta_b = \emptyset$

$\Rightarrow a \parallel b$

Level 1 Level 2

- **Parallelize (cd + ē):**

- **Parallelize (cd)**

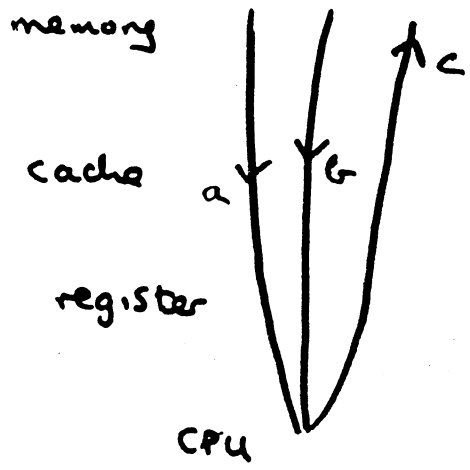


$\Delta_c \cap \Delta_d \neq \emptyset$

- **Parallelize (ē)** ....
- **Parallelize (f)** ....

Fine Granularity and Cache/Register Management

This is "the travelling salesman problem with a few salesmen"...



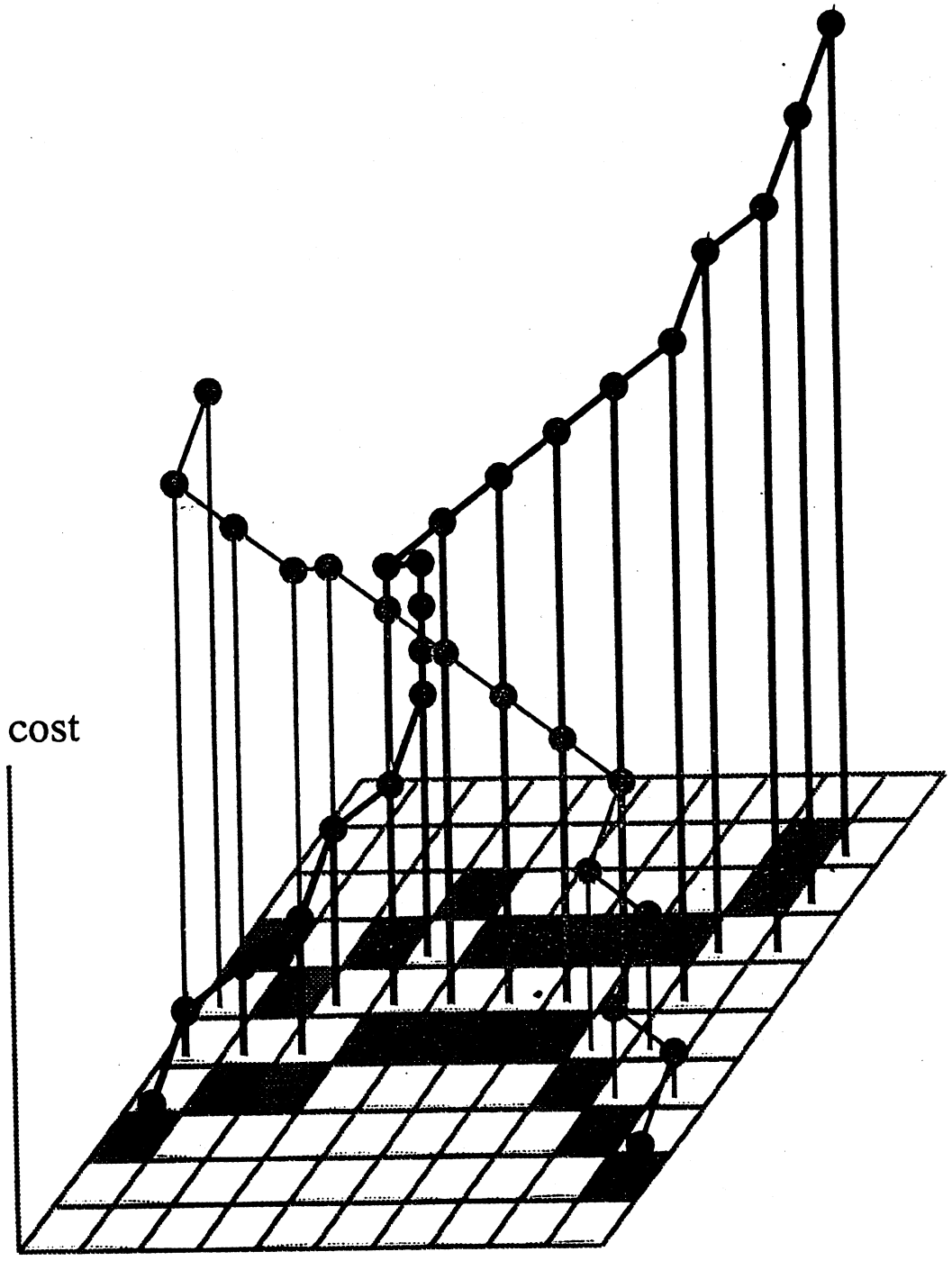
$c = a + b$

Kohler looked at neural networks

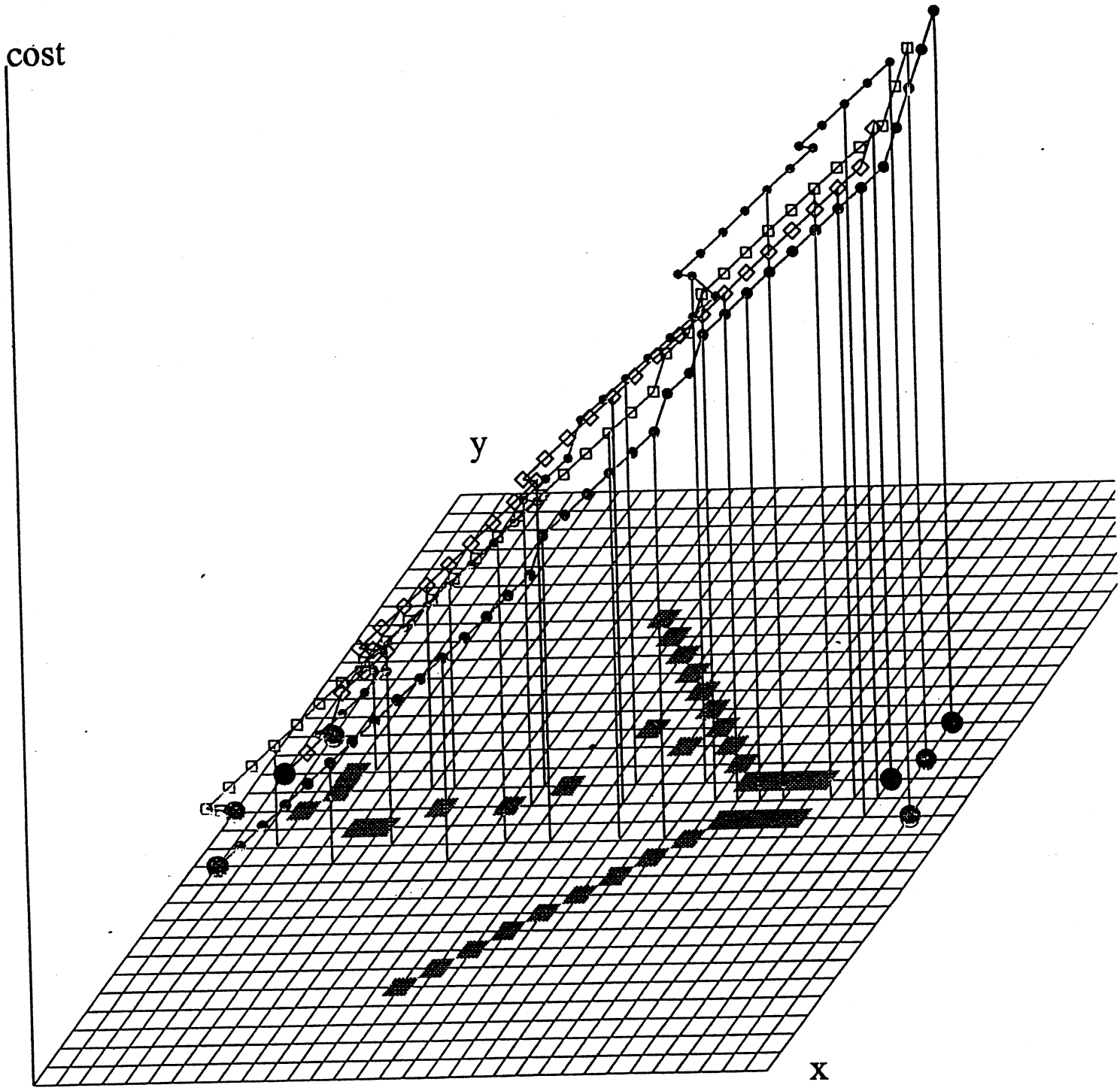
Elastic nets seem promising

We only have experience with navigation problems. Note analogy of CPU and narrow pass in examples

It will require a lot more work to see if we can use these physical optimization methods for compile-time or run time instruction scheduling

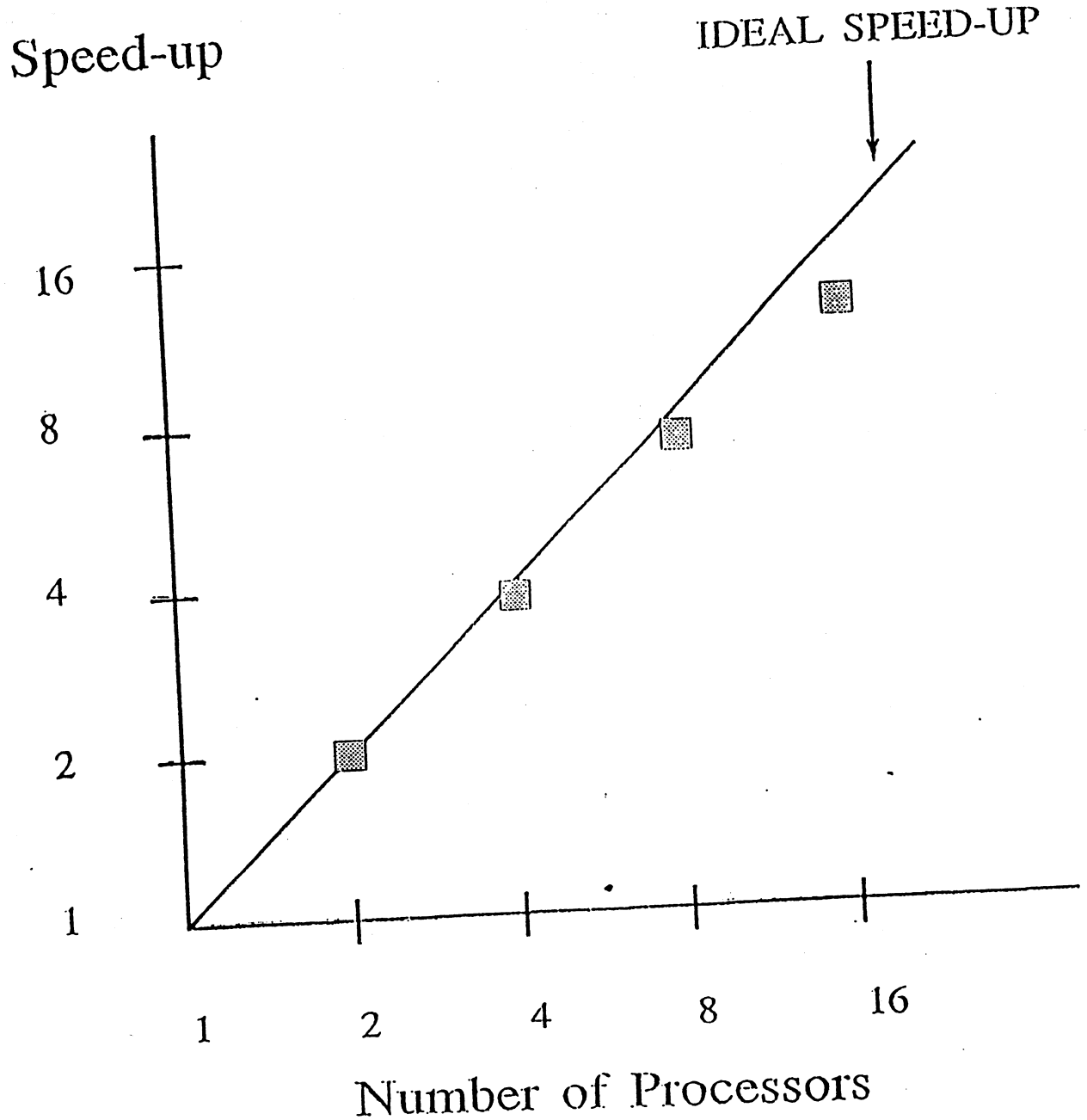


The two-vehicle navigator solution
for a conflict imposing terrain



Four paths in the cost-terrain space calculated by the neural net

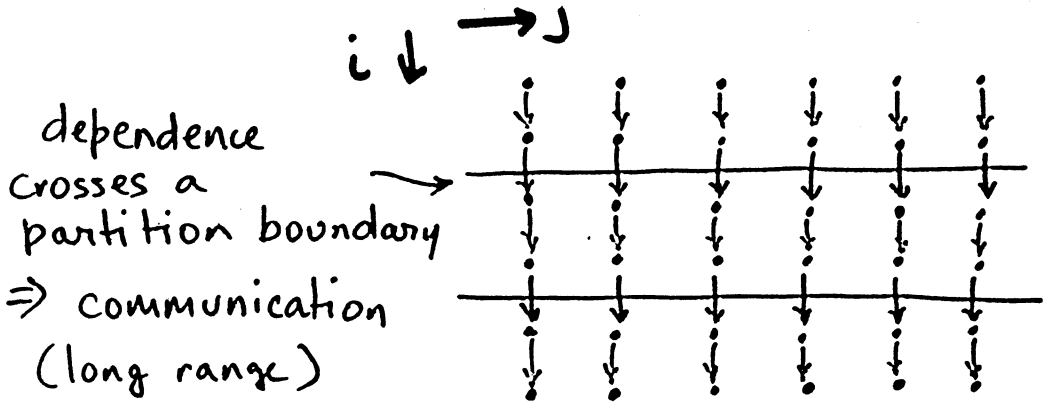
Speed-up for 4 vehicles running on 16-node Meiko

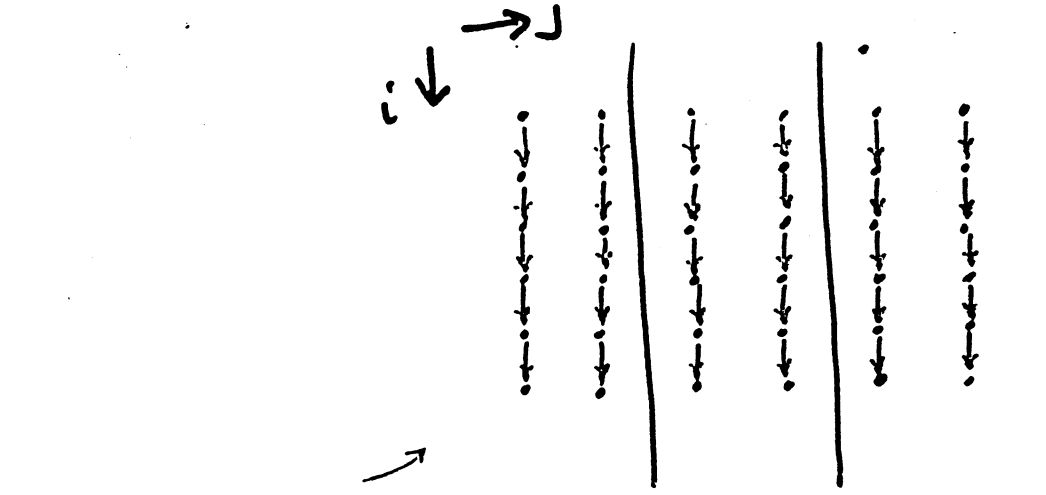


Kremer - Kennedy - BalaSundaram - Fox

Example

```
do i = 1, n
  do j = 1, n
    A(i,j) = f(A(i,j), A(i-1,j))
  enddo
enddo
```



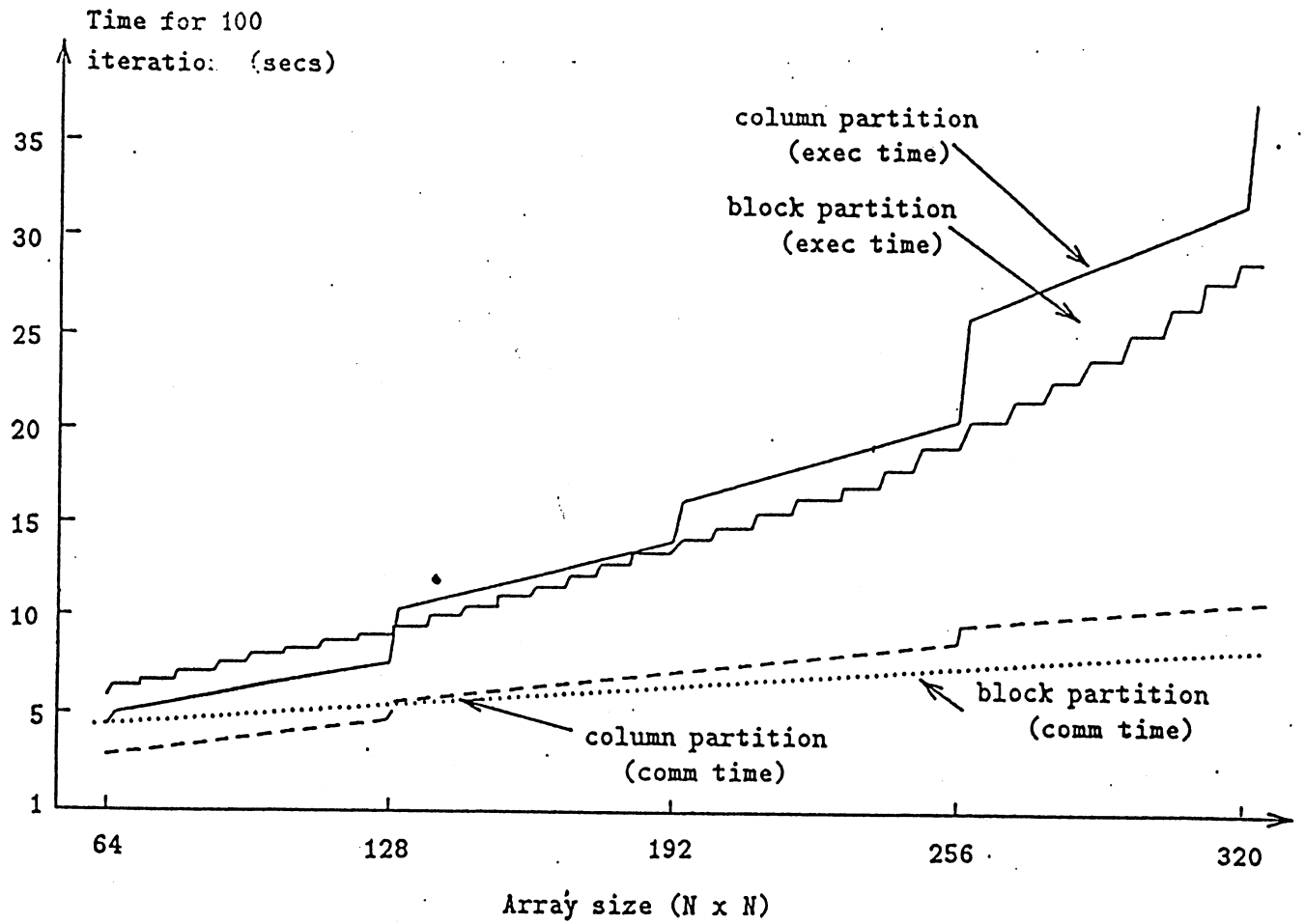


Effect of data partitioning scheme on performance

```
do k = 1, ncycles
   do j = 1, n
      do i = 2, n-2
         A(i,j) = f (B(i-1, j), B(i+1, j))
      enddo
   enddo

   do j = 2, n-1
      do i = 2, n-1
         B(i,j) = f'(A(i-1, j), A(i+1, j), A(i, j),
                     A(i, j-1), A(i, j+1))
      enddo
   enddo
enddo
```

f is 4 and f' 10 double precision floating point operations

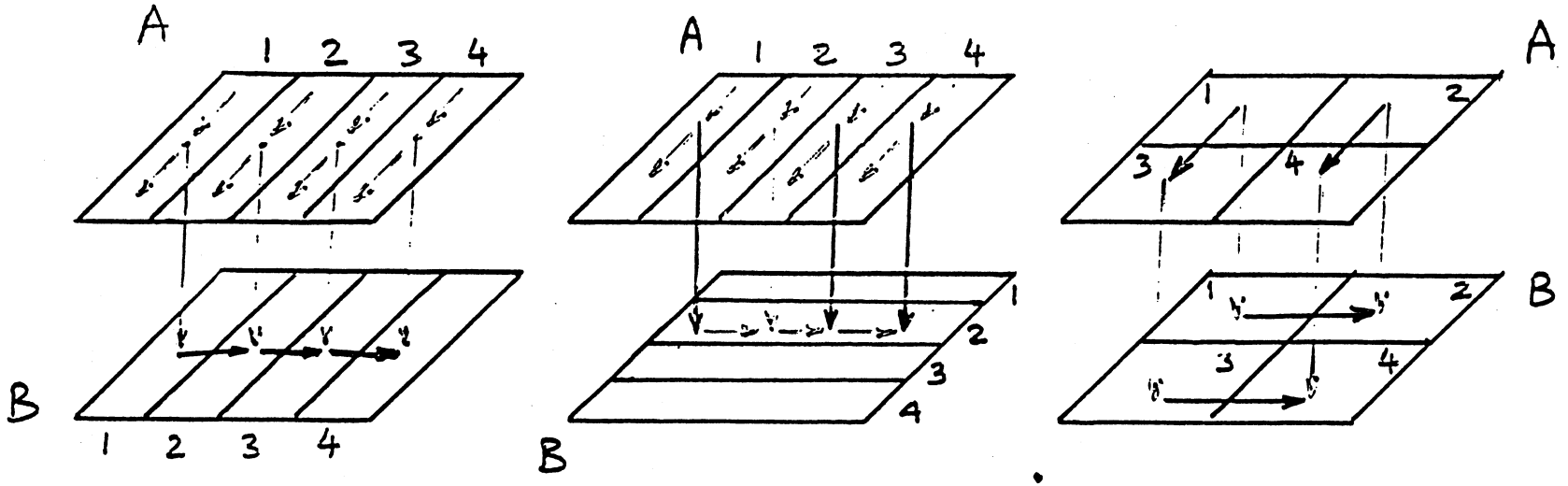


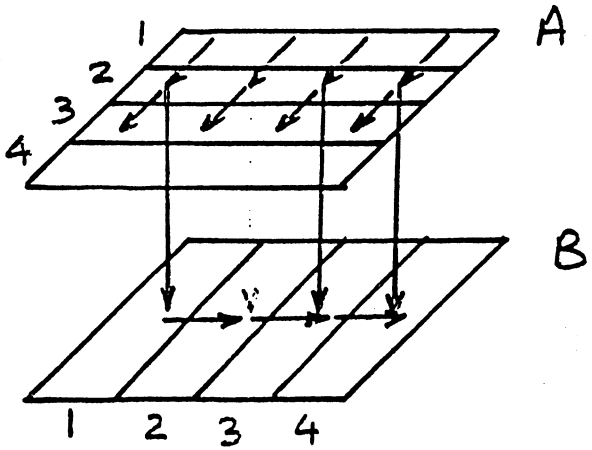
64 node NCUBE

P250

→ Dependence satisfied by [illegible]

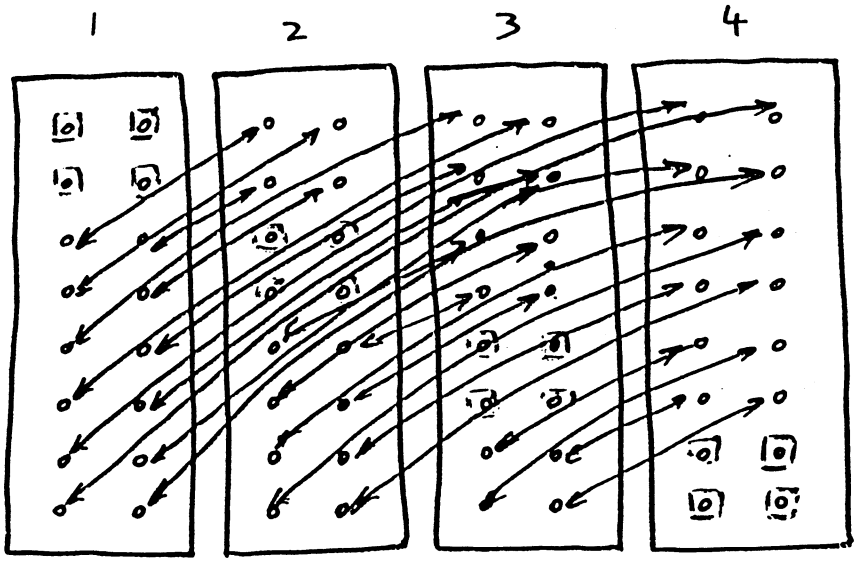
→ Dependence satisfied by communication.





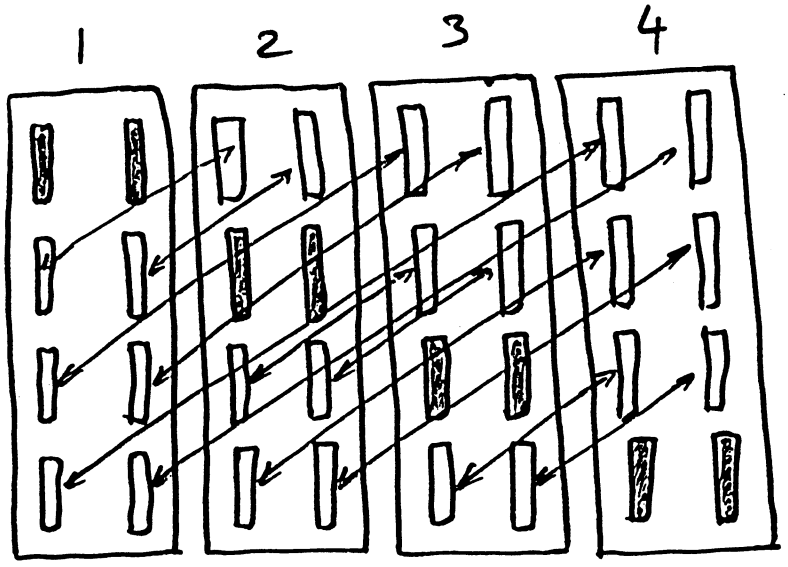
P1

```
do j = 1, n
   do i = 1, n
      A(i,j) = A(i-1,j) * T
      communicate A(i,j)
      B(i,j) = A(i,j) + B(i,j-1) + B(i,j)
   enddo
enddo
```



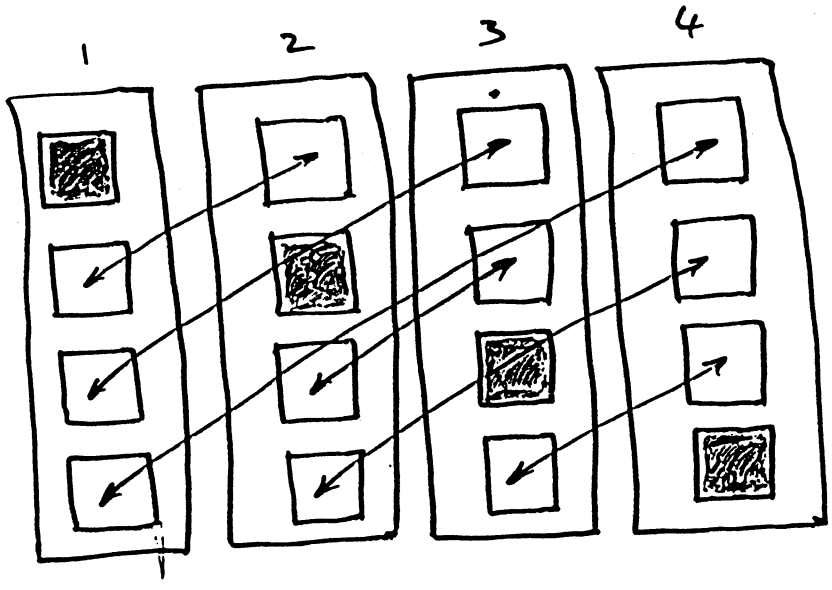
P2 After distributing the i loop

```
do j = 1, n
  do i = 1, n
    A(i,j) = A(i-1, j) * T
  enddo
  communicate A(*, j)
  do i = 1, n
    B(i,j) = A(i,j) + B(i,j-1) + B(i,j)
  enddo
enddo
```



P3 After distributing outer j loop

```
do j = 1, n
  do i = 1, n
    A(i,j) = A(i-1,j) * T
  enddo
enddo
communicate  A(*,*)
do j = 1,n
  do i = 1, n
    B(i,j) = A(i,j) + B(i,j-1) + B(i,j)
  enddo
enddo
```



We can estimate performance of a given decomposition and indeed derive "optimal" decompositions in terms of a model of the machine

Calculation Time of Node — t_{float}, $t_{integer}$

Communication — Latency t_0 + # words. t_{comm}
Bandwidth

There are a lot of variables

Routing Hardware

Can one use >1 channel at a time

Performance of calculational component depends on pipelining, caches etc.

Here we can use neural networks as we aren't violating constraints - just getting non optimal performance

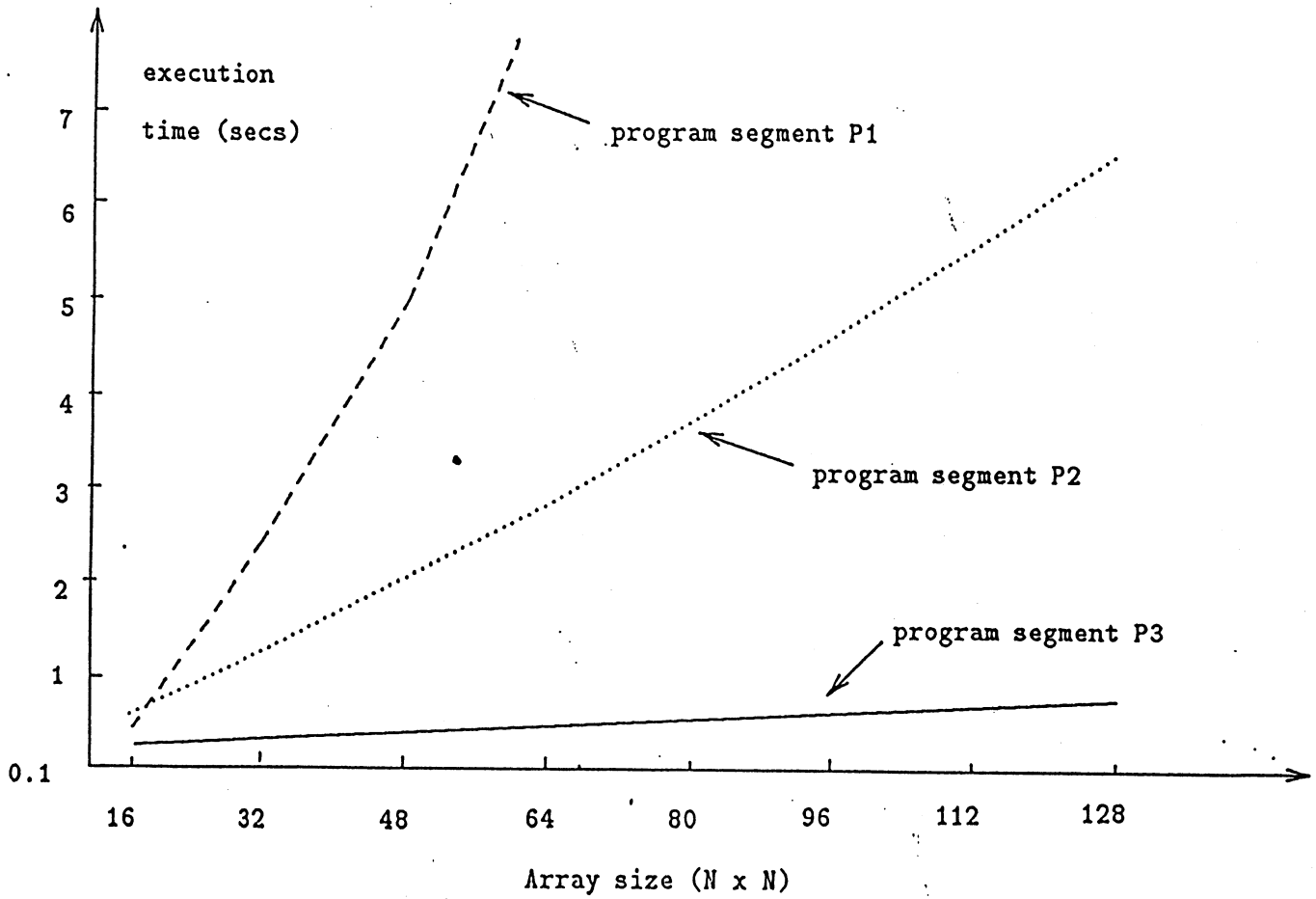
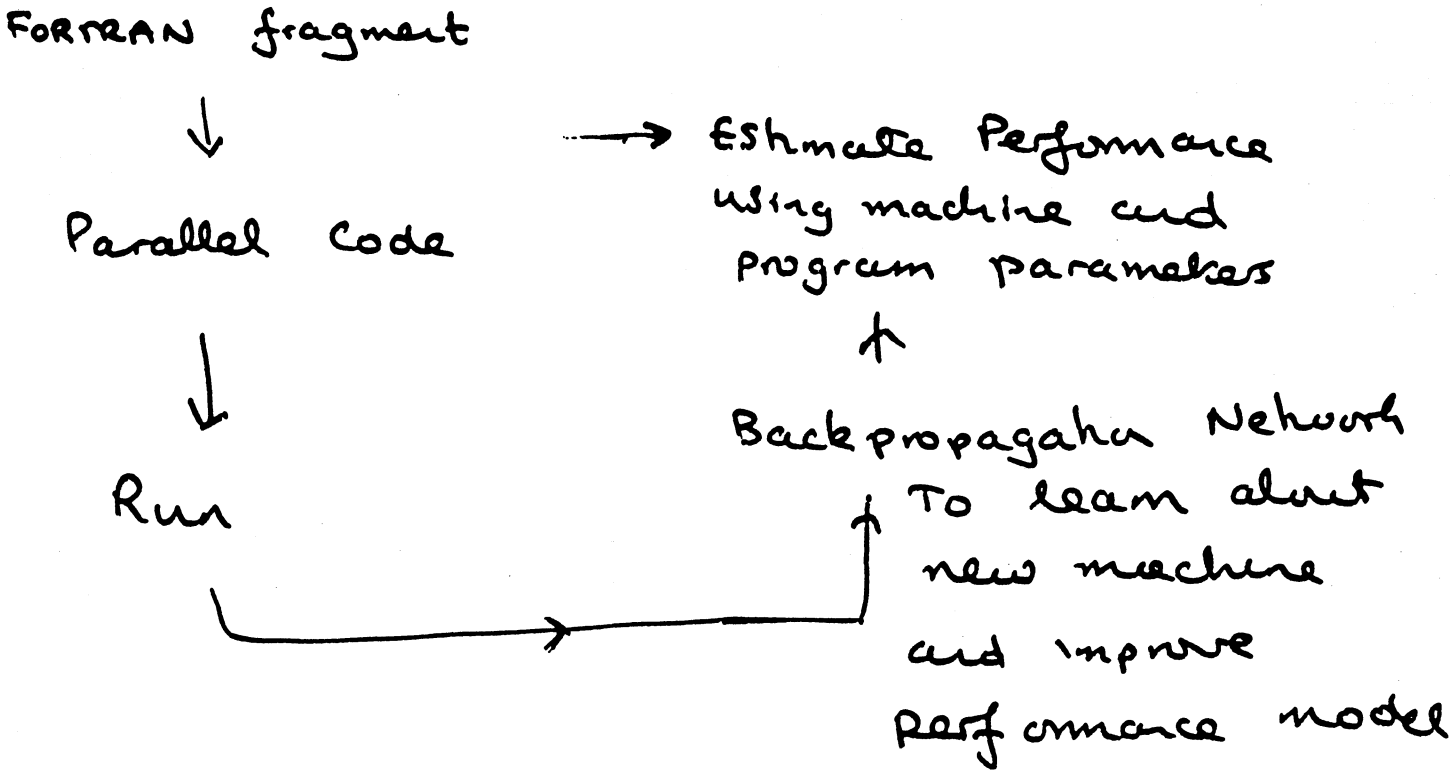


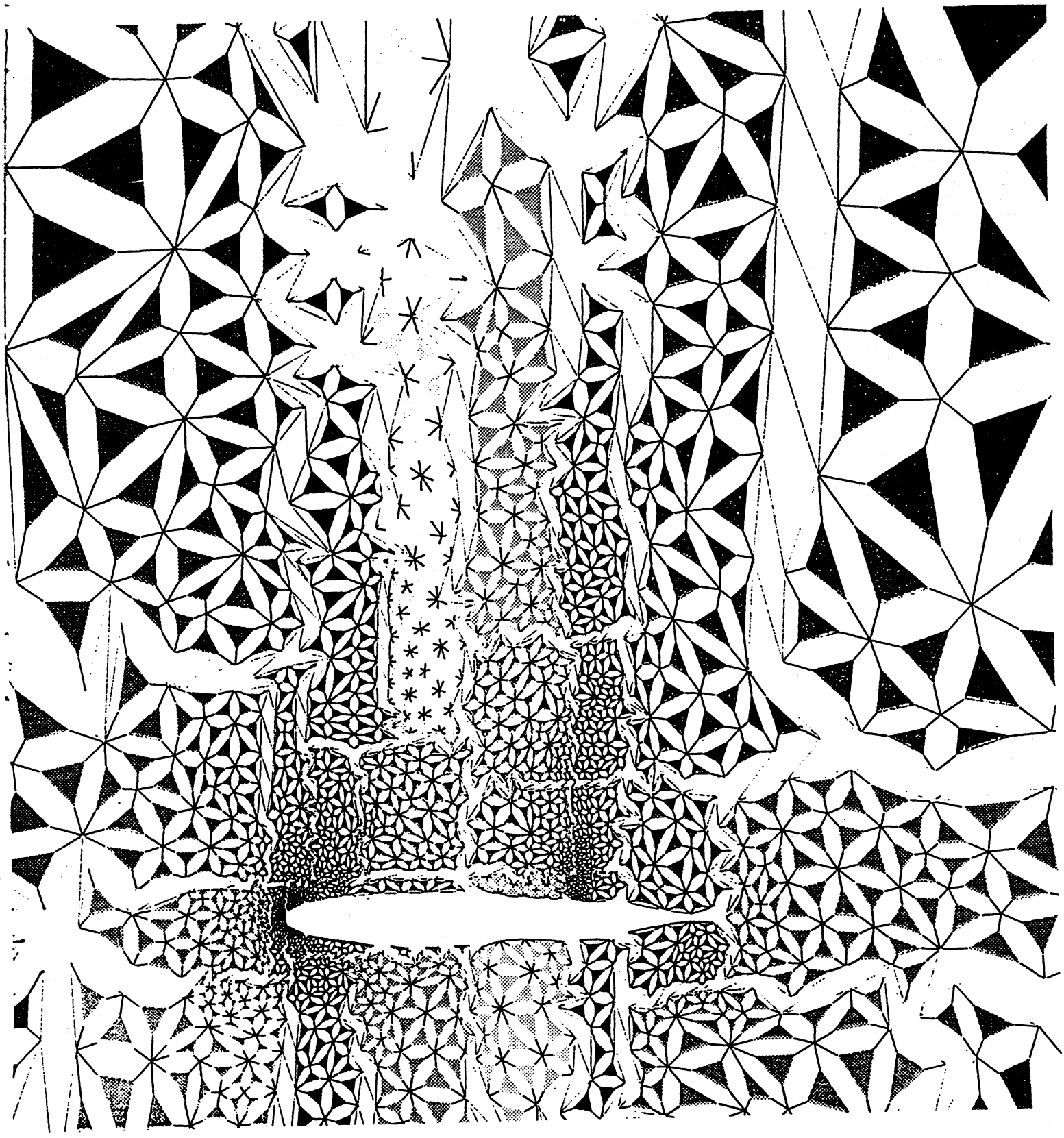
FIGURE 3: Timing results on an NCUBE, using 16 processors.

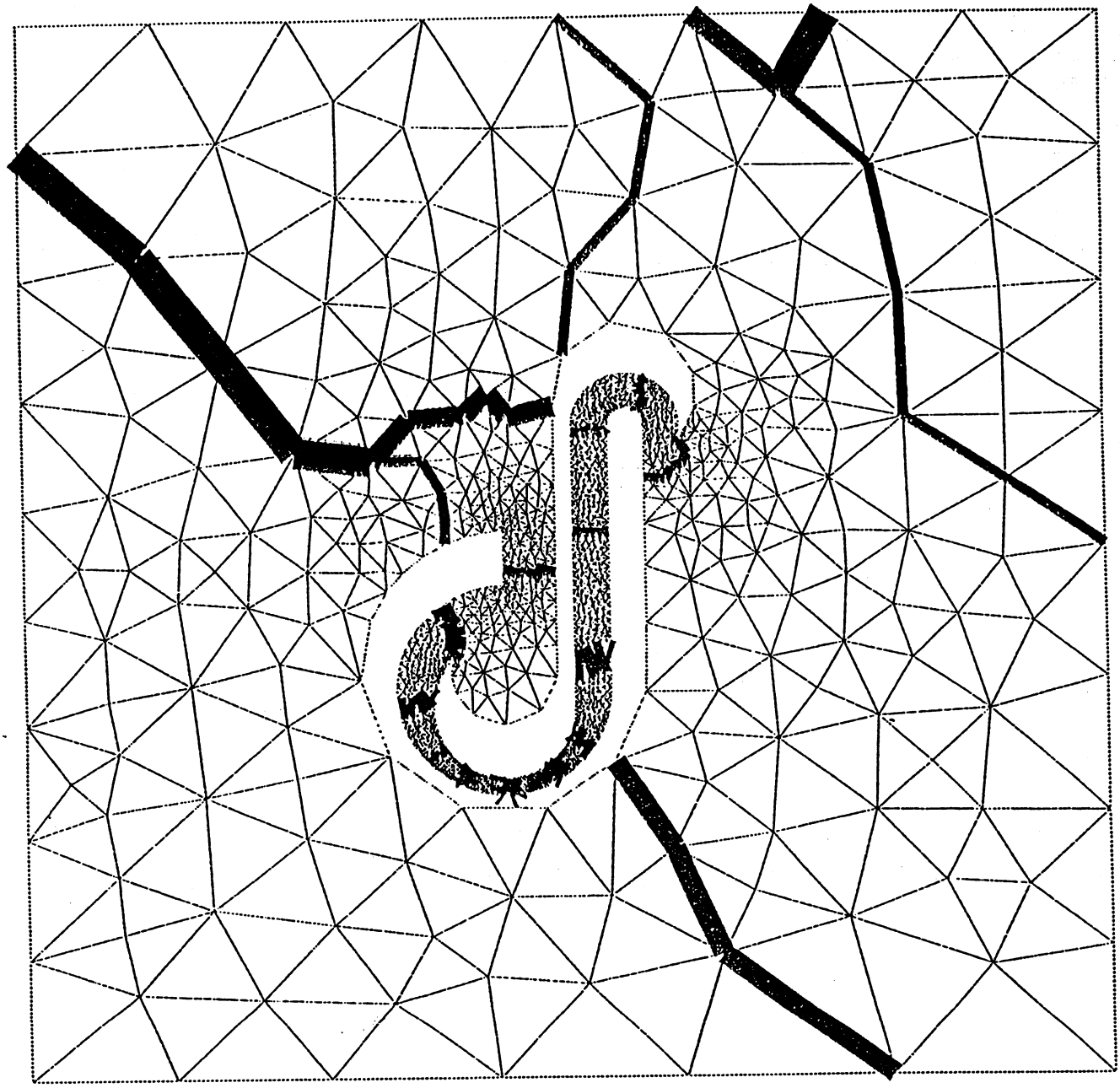


- This network could be taught
  - Initially on a simple "Livermore Loops"
  - In important cases, on a particular code

Here we are using a "learning network" to classify space of decompositions by execution time of that decomposition.

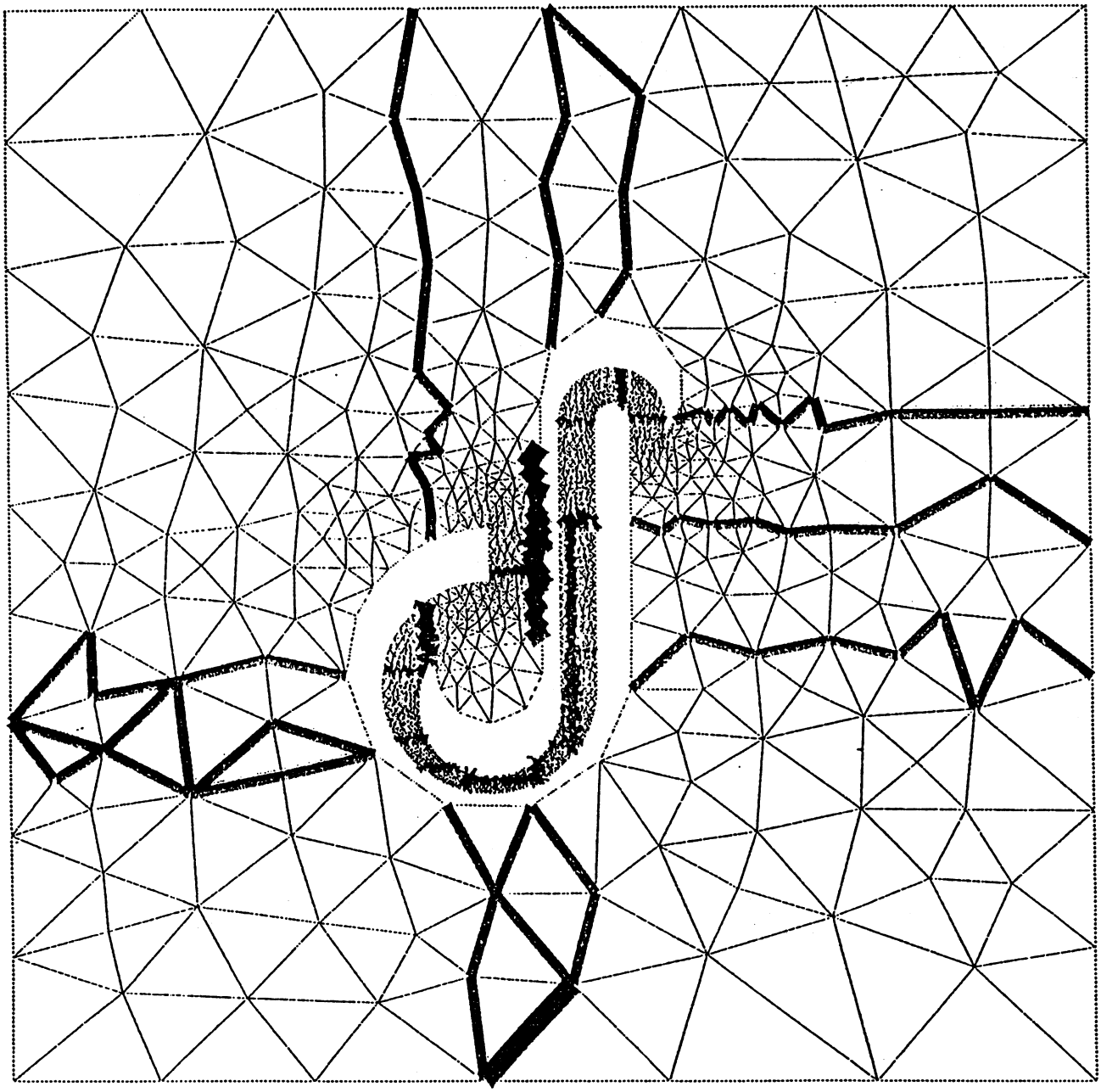
ESpecially for irregular problems, one can use ophmization methods (simulated annealing, neural nets) for compile time or more likely run time decomposition.



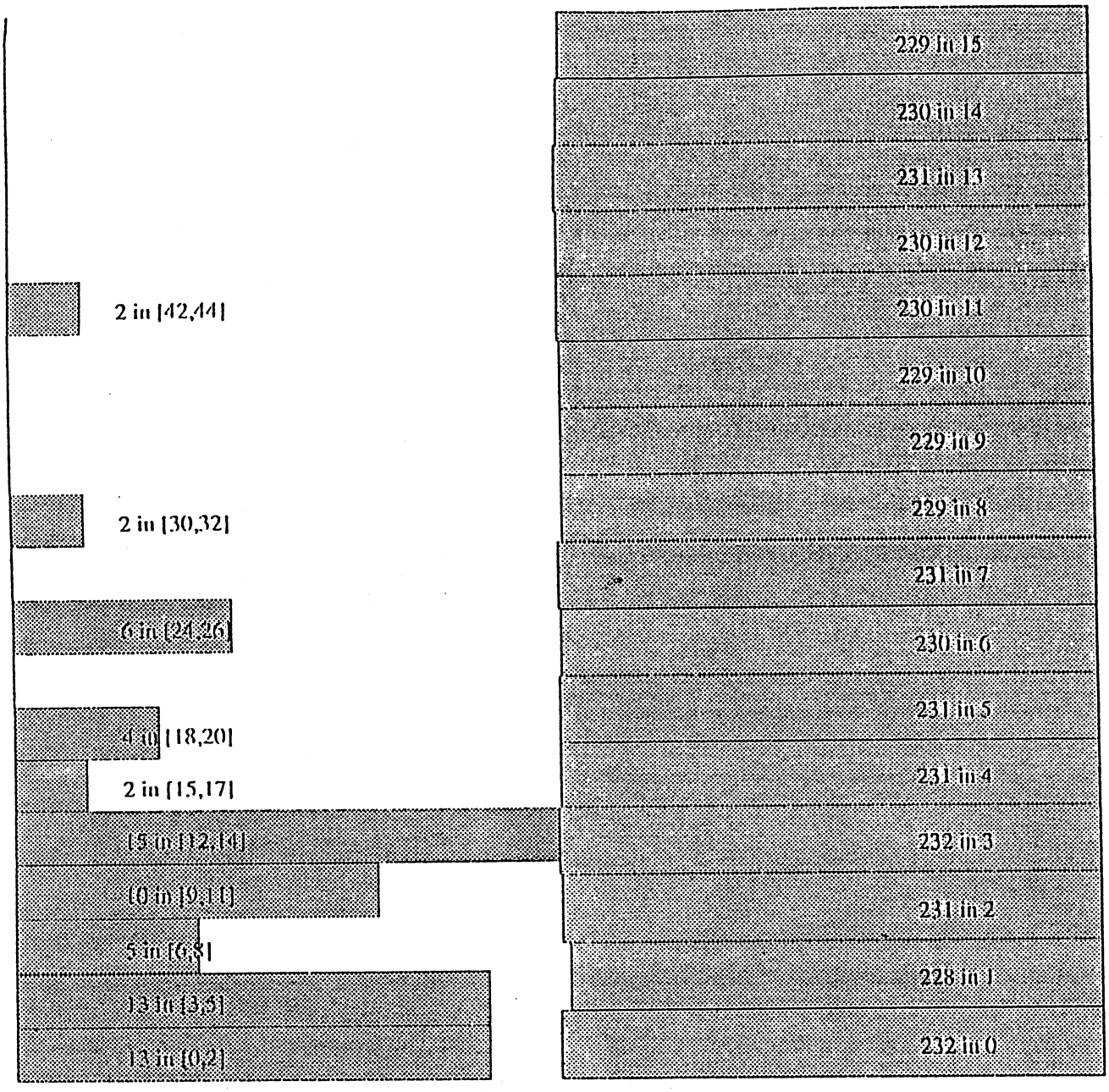


ANNEALING-

Width of line = Hamming distance



ORTHOGONAL RECURSION BISECTION



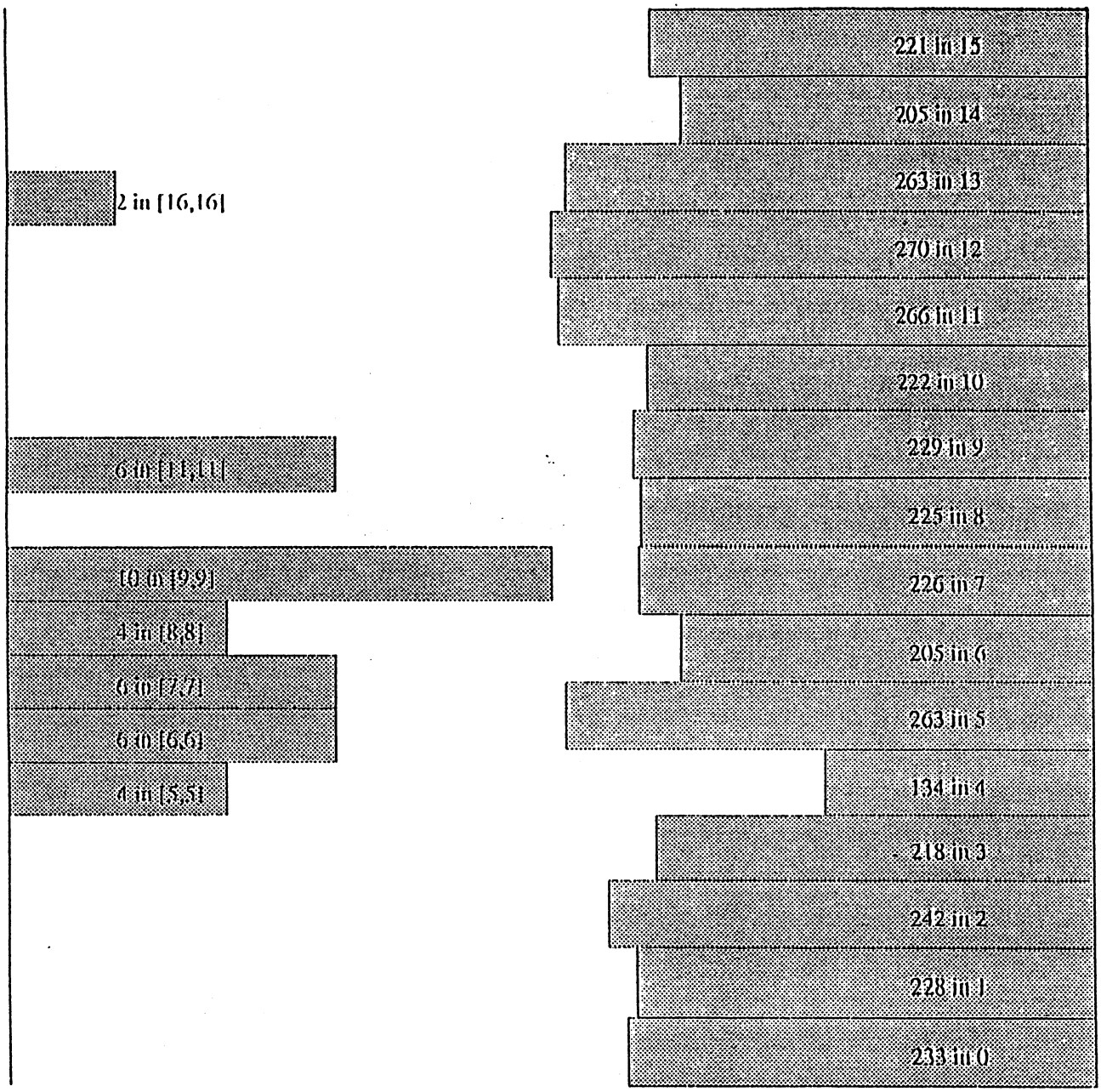
Global Message Histogram

Elmts per Processor

Messages 72, Total size 835

ORTHOGONAL RECURSIVE BISECTION

annealing



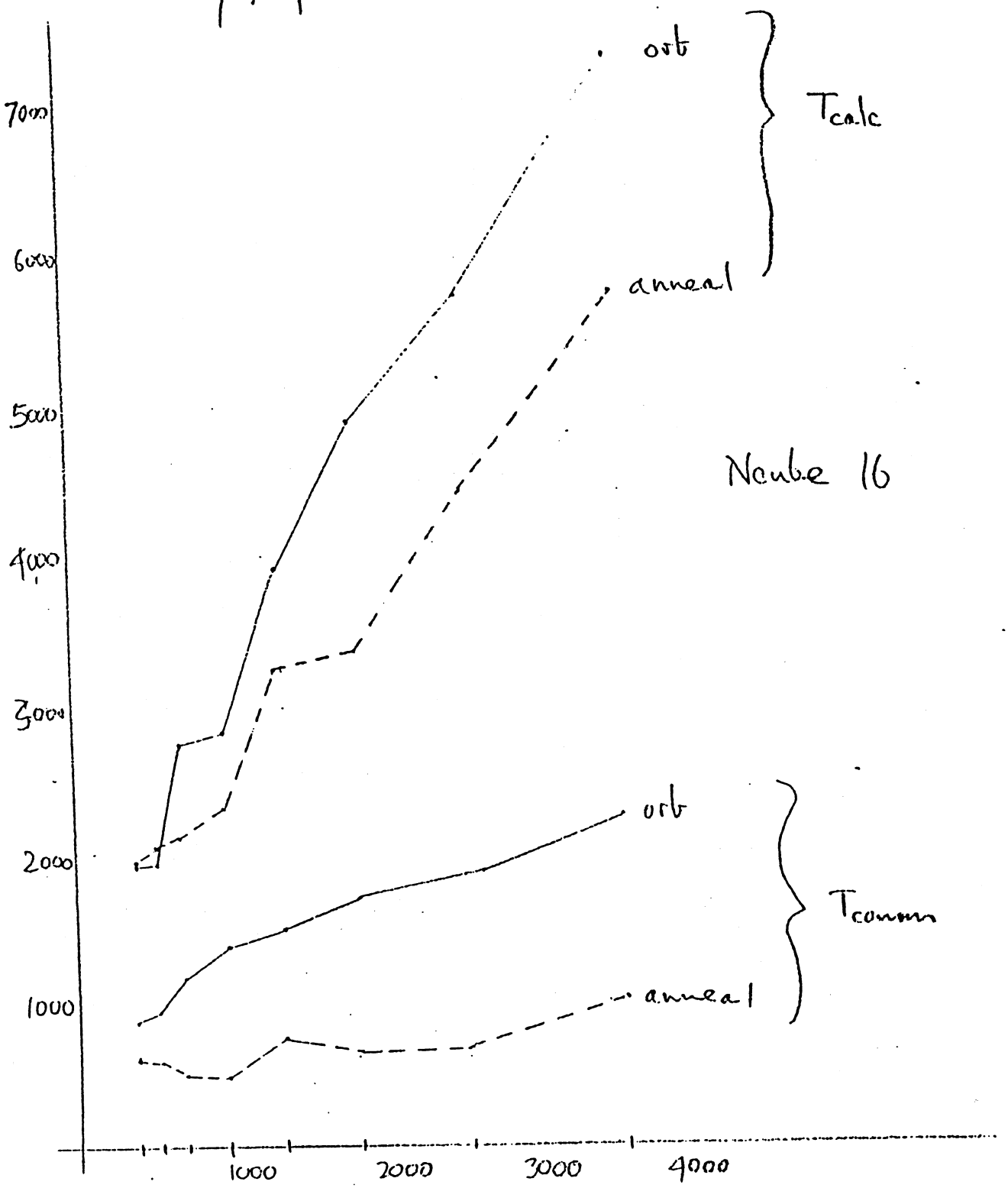
Global Message Histogram

Elmts per Processor

Messages 38, Total size 337

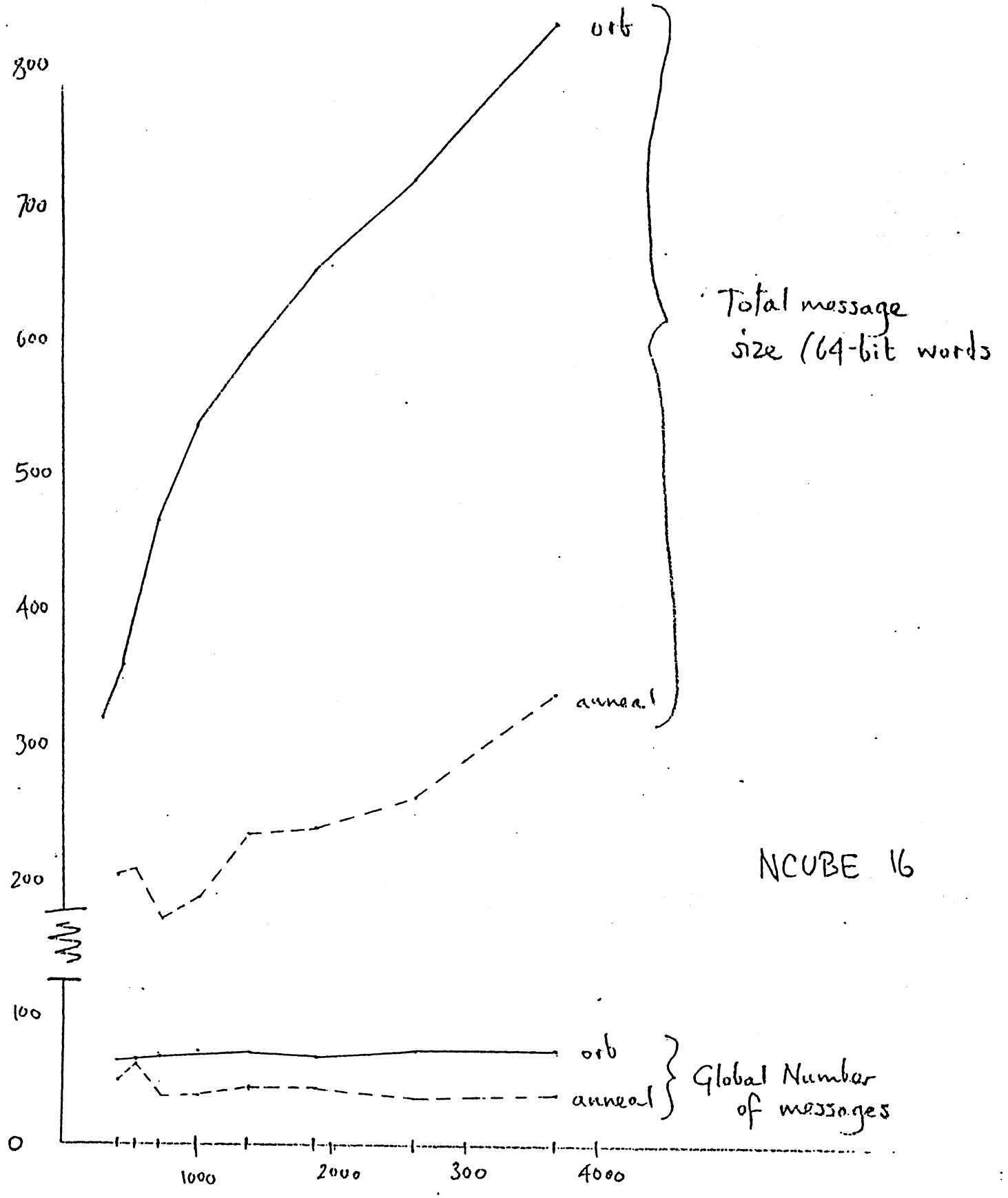
ANNEALING

Time per SOR iteration
(scaled by flop)



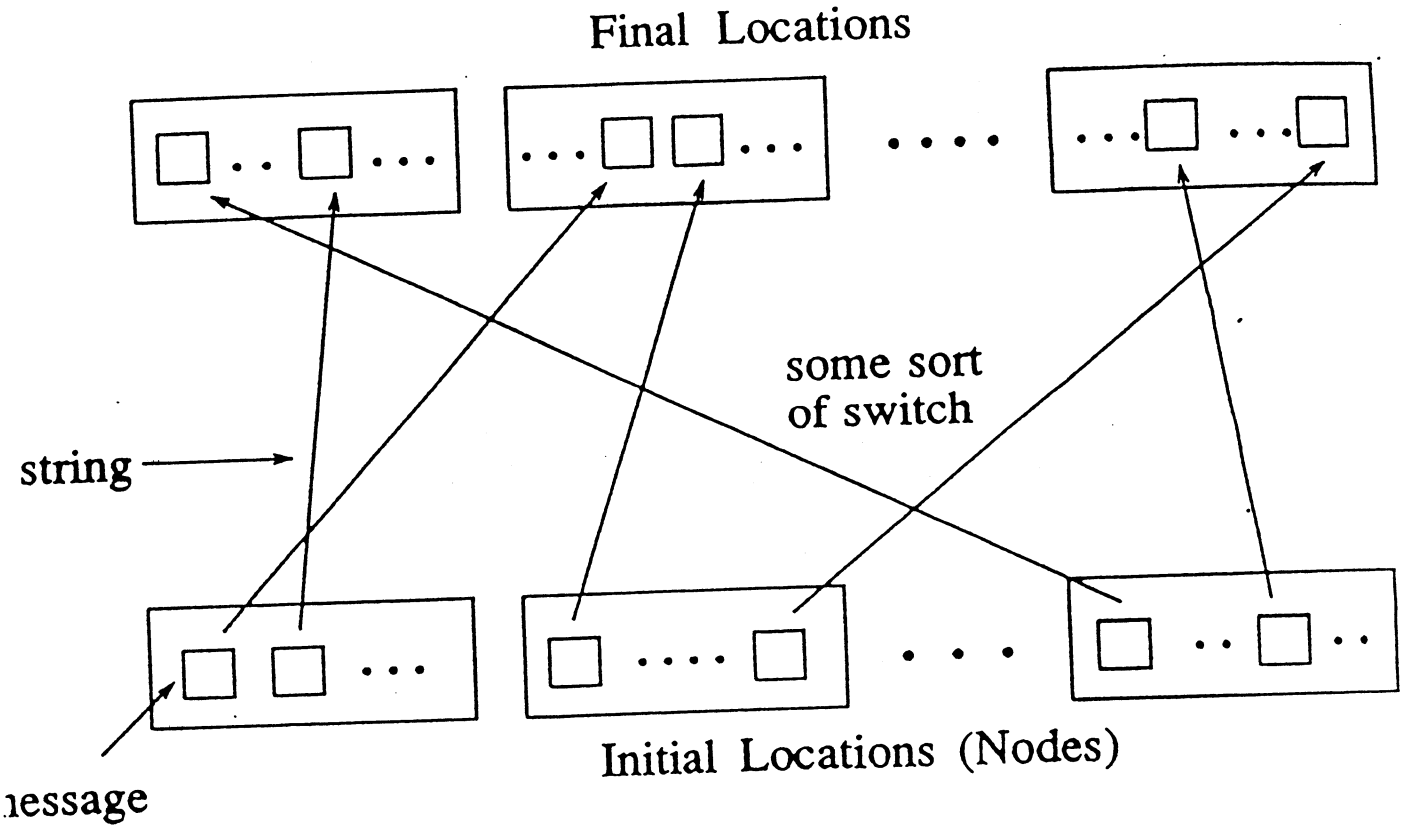
Total Elements

EXECUTION TIME



MESSAGE PERFORMANCE

Dynamic Routing of Messages



Neural_Router dynamically routes messages given current message location and destination

Message Routing on a Parallel Computer

e.g., how to send a set of messages from given source nodes to given destination nodes on a hypercube

- Similar assignment problem but entities (message this time, not grid points) are not fixed or slowly varying but rather move on a fine time scale

  Initial time t_0

  Final time t_1

Goal: Minimize total routing time

$$t_1 - t_0$$

- Now degrees of freedom are not instantaneous positions

$$N(m)$$

node — N; message — m

but rather paths

$$\left\{ N\,(m,\,t) \right\}_{t_0}^{t_1}$$

- Use neural variables as in decomposition problem

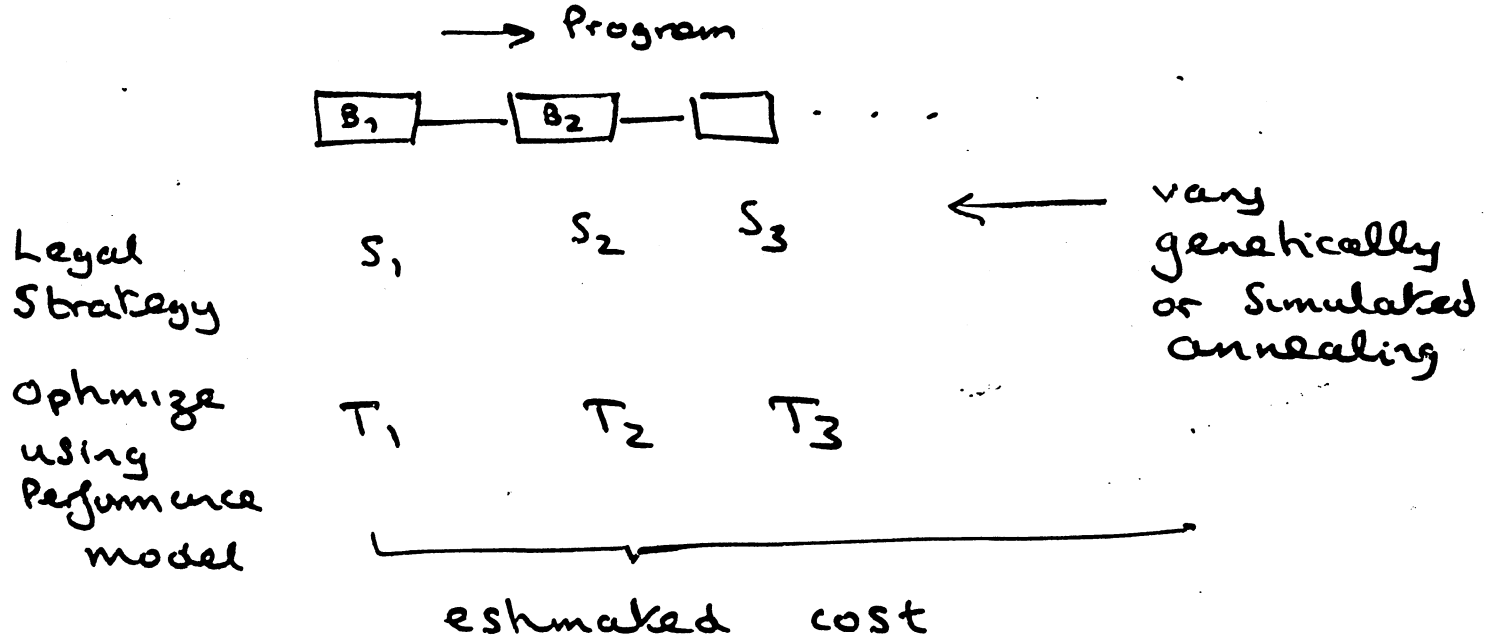
leads to $\eta_k\,(m,t)$

m labels messages

k labels scale level

→ Routing ~ Neural computer

- How do you put together several different blocks - each with its own suite of possible optimization strategies which can clash.
- e.g. Particle in the cell codes go back and forth between particle and grid decompositions
- Here we can use a ~trivially parallel genetic algorithm



- Learning networks can be used to optimize order in which strategies S_i chosen for given i. Must feedback information to compiler!

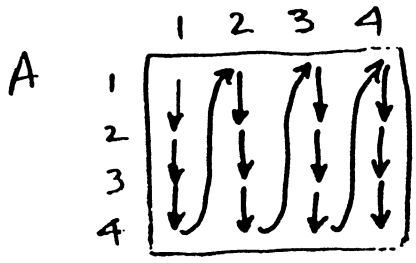
Effect of parallelism improvement transformations on performance

```
      do j = 1, n
parallel
loop  --> do i = 1, n
            A(i,j) = f(A(i-1,j))
          cnddo
        enddo
```

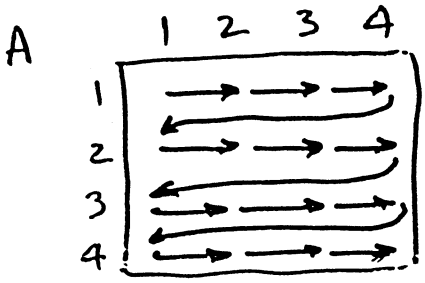
Loop interchange

```
parallel
loop  --> do i = 1, n
            do j = 1, n
              A(i,j) = f(A(i-1,j))
            enddo
          enddo
```

- Assume A is a 4×4 array.
- Let cache size in each processor be 4.



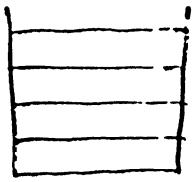
access pattern
before loop interchange



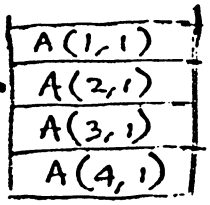
access pattern
after loop interchange

- Arrays are stored in column-major in Fortran.

A(1,1)
cache miss



A(2,1)
hit



A(3,1) hit

⋮

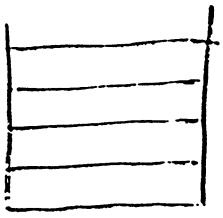
A(1,2) next cache miss

⋮

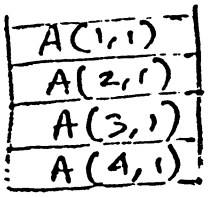
A(1,3) next cache miss

⋮

A(1,1)
cache miss



A(1,2)
cache miss



A(1,3)
cache miss

⋮

⇓

every reference results in a cache miss!

Effect of pipeline utilization improvement transformation on performance

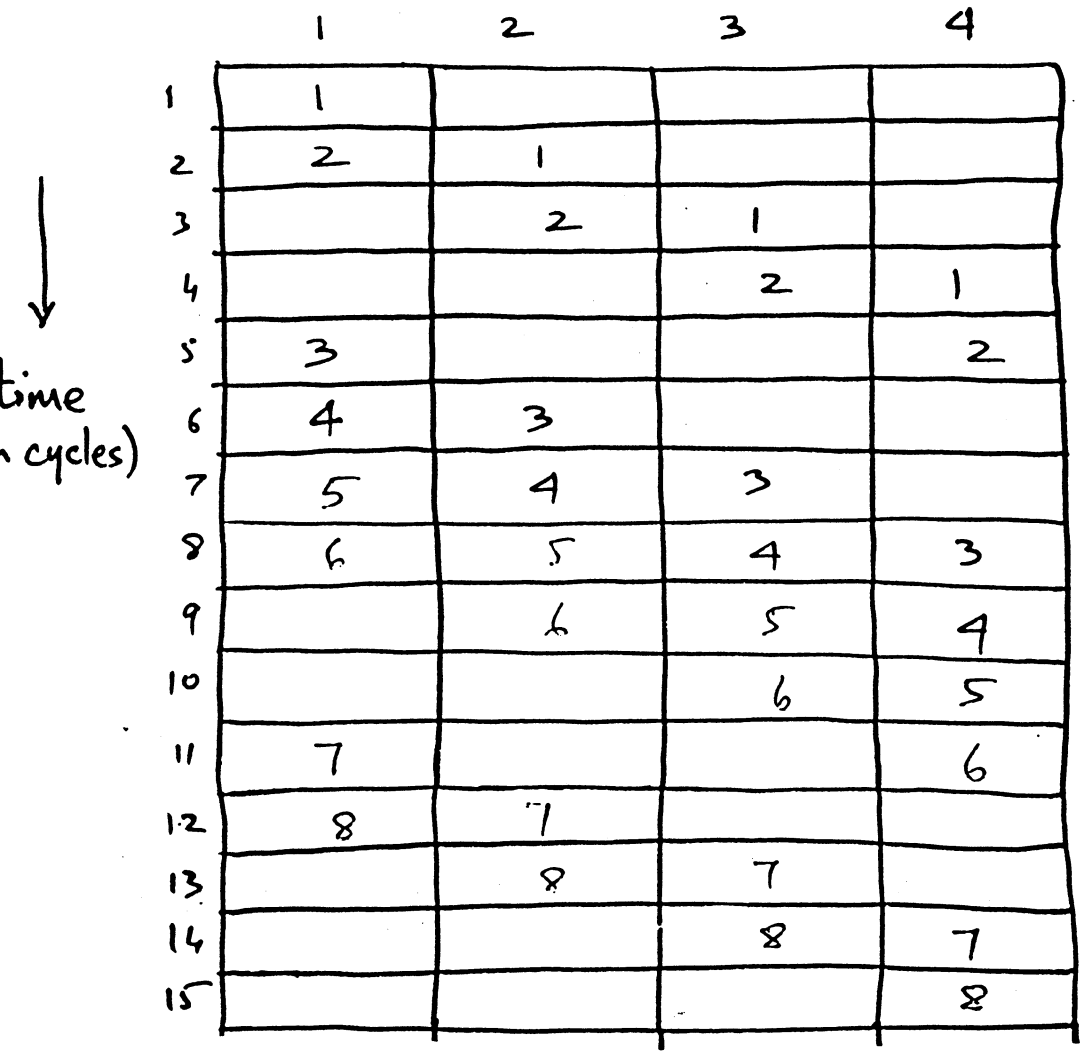
```
      do j = 1, m
        do i = 1, n
S1:       A(i,j) = A(i+1, j) + A(i-1, j)
S2:       B(i,j) = A(i, j-1) + B(i-1, j)
        enddo
      enddo
```

Illustration for $m = n = 2$:

$$
\begin{aligned}
A(1,1) &= A(2,1) + A(0,1) && 1\\
B(1,1) &= A(1,0) + B(0,1) && 2\\
A(2,1) &= A(3,1) + A(1,1) && 3\\
B(2,1) &= A(2,0) + B(1,1) && 4\\
&\vdots\\
A(1,2) &= A(2,2) + A(0,2) && 5\\
B(1,2) &= A(1,1) + B(0,2) && 6\\
A(2,2) &= A(3,2) + A(1,2) && 7\\
B(2,2) &= A(2,1) + B(1,2) && 8
\end{aligned}
$$

- Assume 4 stage pipeline

pipeline stages



Total time = 15 cycles.

Bad pipeline utilization.

After unroll and jam

```
do j = 1, m, 2
   do i = 1, n
      A(i,j) = A(i+1,j) + A(i-1,j)
      B(i,j) = A(i,j-1) + B(i-1,j)
   enddo
   do i = 1, n
      A(i,j+1) = A(i+1,j+1) + A(i-1,j+1)
      B(i,j+1) = A(i,j) + B(i-1,j+1)
   enddo
enddo
```

↓ "Jam"

```
do j = 1, m, 2
   do i = 1, n
      A(i,j) = A(i+1,j) + A(i-1,j)
      B(i,j) = A(i,j-1) + B(i-1,j)
      A(i,j+1) = A(i+1,j+1) + A(i-1,j+1)
      B(i,j+1) = A(i,j) + B(i-1,j+1)
   enddo
enddo
```

$A(1,1) = A(2,1) + A(0,1)$ 1

$B(1,1) = A(1,0) + B(0,1)$ 2

$A(1,2) = A(2,2) + A(0,2)$ 5

$B(1,2) = A(1,1) + B(0,2)$ 6

$A(2,1) = A(3,1) + A(1,1)$ 3

$B(2,1) = A(2,0) + B(1,1)$ 4

$A(2,2) = A(3,2) + A(1,2)$ 7

$B(2,2) = A(3,1) + B(1,2)$ 8

pipeline stages

time (in cycles)	1	2	3	4
1	1			
2	2	1		
3	5	2	1	
4		5	2	1
5	6		5	2
6	3	6		5
7	4	3	6	
8	7	4	3	6
9		7	4	3
10	8		7	4
11		8		7
12			8	
13				8

Total time = 13 cycles

⇒ Better pipeline utilization

An example of iterative refinement

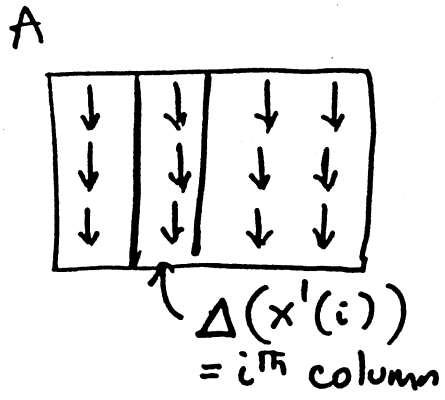
Program P

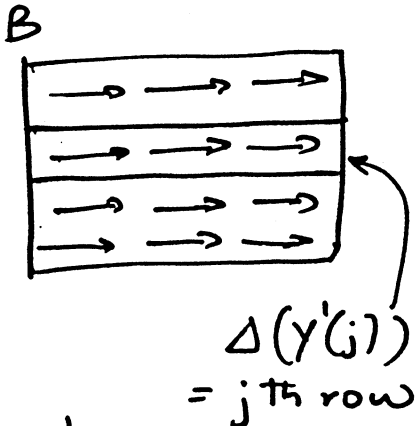
P^0

X^1 (for i
  X^2 (for j $A(i,j) = \cdots A(i-1,j) \cdots$
  endfor)
endfor)

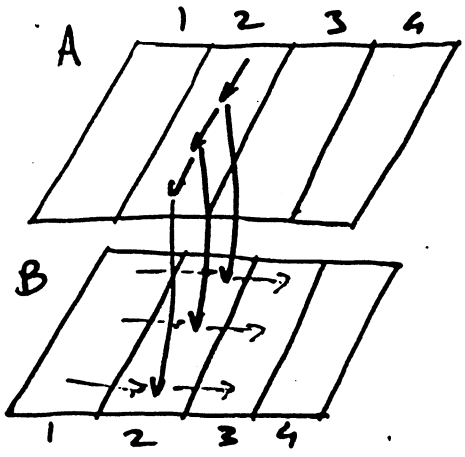
Y^1 (for j
  Y^2 (for i $B(i,j) = A(i,j) + \cdots B(i,j-1) \cdots$
  endfor)
endfor)

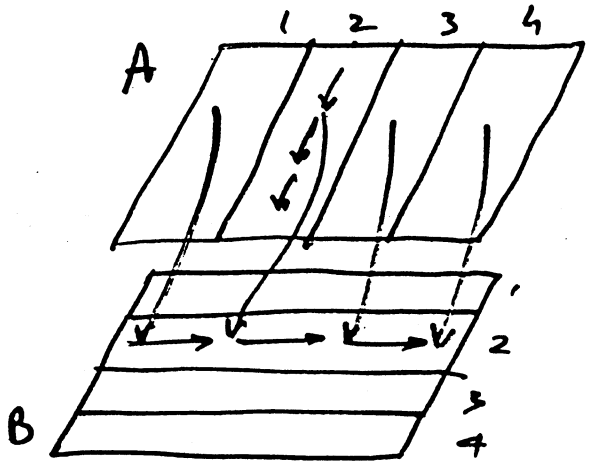
end





Target Machine: MIMD hypercube





⟶ dependence satisfied by communication

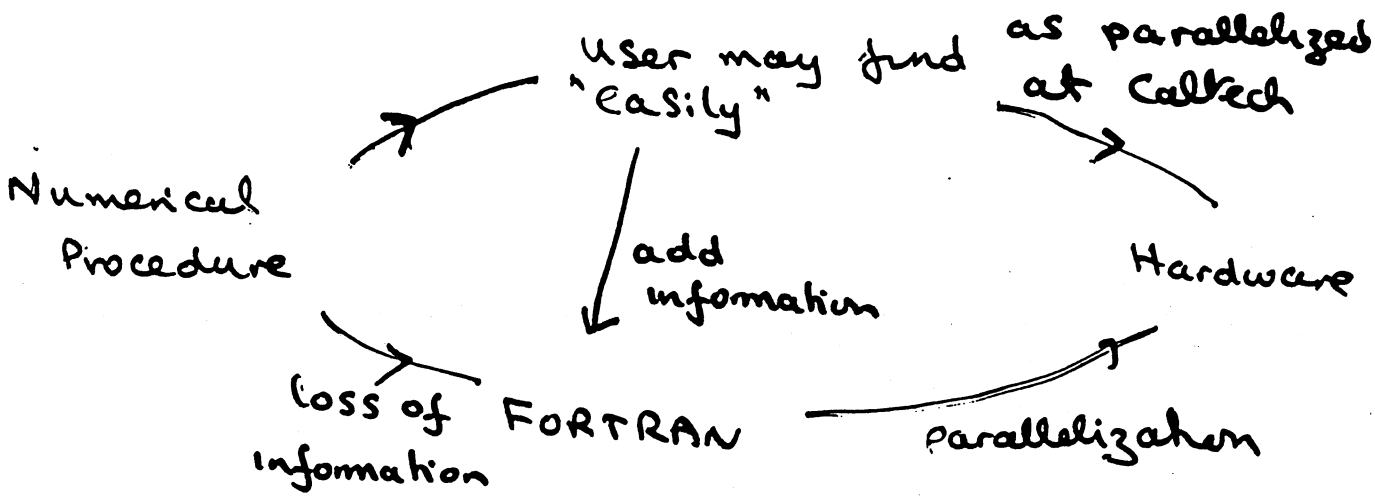
→ dependence satisfied locally

Where does the user fit in?

Each level of the map

problem → hardware

obscures / throws away infomation that is useful e.g. structure of finite element mesh "hidden" in values of pointers.



Currently user helps directly with decomposition but this may not be correct in the long run