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

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

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

Problem
  ↓
Model
  ↓
Method
  ↓
High level Software
  ↓
mix of lower level systems; FORTRAN, communication systems, assembly
  ↓
Hardware

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 from electrical engineering (astronomy?)

(Maximum Entropy)

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 normal? - combine methods
e.g. nature

- Genetic algorithms evolve people over long time periods
- 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 $\eta$ label different parts (eye color, interest in physics..) of system.
Physics or Information 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

If labels states in a physical system

use standard Monte Carlo methods

to obtain states at given $\beta$.

Lower $T$ slowly (e.g. $T_k = 0.99 T_{k-1}$)

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

\[
\langle \eta \rangle = \sum_{\eta} \frac{\eta e^{-\beta f(\eta)}}{Z} \quad \rightarrow \quad \text{desired answer as } \beta \to 0
\]

\[
Z = \sum_{\eta} e^{-\beta f(\eta)}
\]

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

\[
\langle \eta \rangle_T = \frac{\langle \eta \rangle_T}{T}
\]

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

\[ \gamma(a, l, t) = 1 \text{ if variable } a \\
\text{in location } l \\
\text{at time } t \\
= 0 \text{ otherwise} \]

Then one minimizes

\[ f + \text{const. penalty function if } a \]
\text{in two locations at same time} \]

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

Koller discussed in your last meeting

Best in cases when one can violate constraints / allow non-optimality
Simic showed how you could calculate mean field summing over just those $\eta$'s that satisfy syntax constraints. This leads to equations involving new variables that satisfy constraints

$$\langle x(a,t) \rangle = \sum (\langle x(a,t1), \beta \rangle)$$

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

This is elastic net formalism which is a particle dynamics picture e.g. Navigation

![Diagram](image.png)
Travelling Salesman Problem

Each city attracts each particle. Strongest force to nearest particle (these forces are constraints as $\bullet \rightarrow x$ eventually)

Tension minimizes tour length

$\bullet = \text{Particles labelled by time}$

$x = \text{city}$
Progress in Solution of NP Complete Problems

TSP 90 Shoot-out

Blind testing of methods on a challenge 773 City Problem.

- Branch and Cut (Padberg...)
- Exact method: Optimal tour (with guarantee of optimality) in 2 1/2 hours

Physical Optimization: Best implementation is variant of Simulated Annealing + Lin-Kernighan heuristic

Optimal tour in 6 hours (without guarantee). Very good solution (within 2%) in minutes.

(Spare station was common machine)
Learning ("backpropagation")

Neural Networks

- Robust method of "classifying"

  Spaces

  e.g. 64 pixels $\rightarrow$ Net $\rightarrow$ classify as $A \rightarrow Z$

- 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 $\rightarrow$ Neural Net

  whereas choice of algorithm $\rightarrow$ Expert System
<table>
<thead>
<tr>
<th>Application</th>
<th>← MIMD →</th>
<th>← SIMD →</th>
<th>Will Semi-Automatic Parallelizing (FORTRAN) Compiler Work?</th>
</tr>
</thead>
<tbody>
<tr>
<td>QCD</td>
<td>1</td>
<td>6</td>
<td>x</td>
</tr>
<tr>
<td>Continuous Spin (High $T_c$)</td>
<td>1</td>
<td>6</td>
<td>x</td>
</tr>
<tr>
<td>Ising/Potts Models</td>
<td>1</td>
<td>32</td>
<td>10</td>
</tr>
<tr>
<td>Strings</td>
<td>1</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>Particle Dynamics $O(N \log N)$</td>
<td>1</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>$O(N^2)$</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Astronomical Data Analysis</td>
<td>IBM 3090</td>
<td>NCUBE</td>
<td>?</td>
</tr>
<tr>
<td>2 pulsars in 1989</td>
<td>5 pulsars in 1989</td>
<td>?</td>
<td>x</td>
</tr>
<tr>
<td>Chemical Reactions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$H + H_2$ Scattering</td>
<td>1</td>
<td>1.5</td>
<td>x</td>
</tr>
<tr>
<td>$e^- + CO$ Scattering</td>
<td>1</td>
<td>13</td>
<td>x</td>
</tr>
<tr>
<td>Grain Dynamics</td>
<td>1</td>
<td>6</td>
<td>x</td>
</tr>
<tr>
<td>Plasma Physics</td>
<td>1</td>
<td>2</td>
<td>x</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>1</td>
<td>6</td>
<td>x</td>
</tr>
<tr>
<td>Computer Chess</td>
<td>1</td>
<td>2.5</td>
<td>?</td>
</tr>
<tr>
<td>Multi-target Tracking</td>
<td>1</td>
<td>12</td>
<td>x</td>
</tr>
</tbody>
</table>
When can FORTRAN be parallelized by compiler

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

or "embarrassingly 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 &X
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 \[ \frac{1}{2} N^2 t_{\text{particle}} \]
Clustering \[ (20-50) N \log_2 N t_{\text{particle}} \] best when \[ N \geq 1000 \]

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 boxing, presented for simplicity in two dimensions. A system of particles is shown, and the recursive subdivision of space induced by these particles.
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 performance model that is based on the target machine's parameters.

- Goal is to maximize the objective function.
  \( \Rightarrow \) 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.

\( (A) \Rightarrow \)
\[
\begin{align*}
&\text{Do } I = 1, N1 \\
&\quad \text{Do } J = 1, N2 \\
&\quad B(I, J) = (A(I-1, J) + A(I+1, J) \\
&\quad \quad + A(I, J-1) + A(I, J+1) \times 2.5)
\end{align*}
\]
\( (B) \Rightarrow \)
\[
\text{Enddo}
\]

- Data parallelism for loosely synchronous (regular or irregular) problems \( \Rightarrow \) collective communication
  i.e. at \((A)\) not \((B)\).
Collective communication notes 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 Johnson

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

\[ G := X \]
\[ X := \prod_{i} L(\text{Ci}) \]
\[ \Delta(x) = \bigcup_{i} \Delta(\text{Ci}) \]

\[ := \text{Loop} \]
\[ L(x) = \overrightarrow{L(a)} \]
\[ \Delta(x) = \uparrow \Delta(a) \]

\[ := \text{IF} \]
\[ L(x) = L(a) + L(b) \]
\[ \Delta(x) = \Delta(a) \cup \Delta(b) \]

\[ := \text{PARALLEL} \]
\[ L(x) = \bigcup_{i} L(\text{Ci}) \]
\[ \Delta(x) = \bigcup_{i} \Delta(\text{Ci}) \]

\[ := \text{call S} \]
\[ L(x) = \{ L(a) \} \]
\[ \Delta(x) = \uparrow \Delta(s) \]

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 ($G$):

\[ ab(c + \overline{e}) \]

Level 0

Parallelize ($ab$):

\[ ab \]

Level 1

\[ a \quad \Delta a \quad b \quad \Delta b \quad \Delta a \cap \Delta b = \emptyset \quad \Rightarrow \quad \text{all } b \]

Level 2

Parallelize ($cd + \overline{e}$):

\[ cd + \overline{e} \]

\[ cd \quad \overline{e} \]

\[ \Delta cd \quad \Delta e \]

Parallelize ($cd$):

\[ cd \]

\[ c \quad \Delta c \quad \Delta c \cap \Delta d \neq \emptyset \]

Parallelize ($\overline{e}$) ...

Parallelize ($f$) ...
Fine granularity and Cache/Register Management

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

\[ c = a + b \]


We only have experience with navigation problems. Note analogy of CPU and memory 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

Speed-up

Number of Processors

Ideal Speed-Up
Example

\[
\begin{align*}
\text{do } i &= 1, n \\
\text{do } j &= 1, n \\
A(i,j) &= f(A(i,j), A(c-1,j)) \\
\text{enddo} \\
\text{enddo}
\end{align*}
\]

dependence crosses a partition boundary
\Rightarrow \text{ communication (long range)}

\rightarrow \text{ partition A row-wise}

\rightarrow \text{ partition A column-wise}

\Rightarrow \text{ no dependences cross partition boundaries}
\Rightarrow \text{ no communication necessary longrange}
Effect of data partitioning scheme on performance

do $k = 1, \text{ncycles}$

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

$f$ is 4 and $f'$ 10 double precision floating point operations
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
After distributing the i loop

\[
do \ j = 1, n \\
\quad do \ i = 1, n \\
\qquad A(i, j) = A(i-1, j) * T \\
\qquad enddo \\
\qquad communicate A(*, j) \\
\quad do \ i = 1, n \\
\qquad B(i, j) = A(i, j) + B(i, j-1) + B(i, j) \\
\qquad enddo \\
\qquad enddo
\]
After distributing outer j loop

\[
\begin{align*}
&\text{do } j = 1, n \\
&\quad \text{do } i = 1, n \\
&\quad \quad A(i,j) = A(i-1,j) \times T \\
&\quad \text{enddo} \\
&\text{enddo} \\
&\text{communicate } A(\ast, \ast) \\
&\text{do } j = 1, n \\
&\quad \text{do } i = 1, n \\
&\quad \quad B(i,j) = A(i,j) + B(i,j-1) + B(i,j) \\
&\quad \text{enddo} \\
&\text{enddo}
\end{align*}
\]
We can estimate performance of a given decomposition and indeed derive "optimal" decompositions in terms of a model of the machine.

\[
\text{Calculation Time of Node} \quad \text{t}_{\text{float}}
\]
\[
\text{Communication Latency} \quad \text{t}_{\text{comm}}
\]
\[
\text{Bandwidth} \quad \text{t}_{\text{comm}}
\]

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 near optimal performance.
Figure 3: Timing results on an NCUBE, using 16 processors.
FORTRAN fragment

↓

Parallel Code

↓

Run

→ Estimate Performance using machine and program parameters
↑

Backpropagation Network

↑ To learn about new machine

and improve performance model

This network could be trained

- Initially on a sample "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 optimization 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
Time per SOR iteration (scaled by flop)

orb

T_{calc}

anneal

Neube 16

orb

T_{comm}

anneal

Total Elements

Execution Time
Dynamic Routing of Messages

Final Locations

Initial Locations (Nodes)

string

some sort of switch

message

*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 \ gradient\ message

  but rather paths

  \[
  \begin{cases}
  N(m, t) \quad \text{at } t_1 \\
  \quad \text{at } t_0
  \end{cases}
  \]

- Use neural variables as in decomposition problem
  
  leads to $\eta_k(m,t)$

  $m$ labels messages

  $k$ labels scale level

\[\Rightarrow \text{Routing on Neural Computer}\]
• 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

\[ \text{Program} \]

Legal Strategy

\[ S_1, S_2, S_3 \]

Optimize using Performance model

\[ T_1, T_2, T_3 \]

Estimated cost

• 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

\[
\begin{align*}
&\text{do } j = 1, n \\
&\text{parallel loop} \rightarrow \text{do } i = 1, n \\
&\quad \quad A(i, j) = f(A(i-1, j)) \\
&\quad \text{enddo} \\
&\text{enddo} \\
&\text{Parallel loop} \rightarrow \text{do } i = 1, n \\
&\quad \text{do } j = 1, n \\
&\quad \quad A(i, j) = f(A(i-1, j)) \\
&\quad \text{enddo} \\
&\text{enddo}
\end{align*}
\]
Assume $A$ is a $4 \times 4$ array.

- Let cache size in each processor be 4.

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

Downward every reference results in a cache miss!
Effect of pipeline utilization improvement transformation on performance

\[
\text{do } j = 1, m \\
\qquad \text{do } i = 1, n \\
S_1: \quad A(i,j) = A(i+1,j) + A(i-1,j) \\
S_2: \quad B(i,j) = A(i,j-1) + B(i-1,j) \\
\text{enddo} \\
\text{enddo}
\]

Illustration for \( m = n = 2 \):

\[
\begin{align*}
A(1,1) &= A(2,1) + A(0,1) & 1 \\
B(1,1) &= A(1,0) + B(0,1) & 2 \\
A(2,1) &= A(3,1) + A(1,1) & 3 \\
B(2,1) &= A(2,0) + B(1,1) & 4 \\
\vdots \\
A(1,2) &= A(2,2) + A(0,2) & 5 \\
B(1,2) &= A(1,1) + B(0,2) & 6 \\
A(2,2) &= A(3,2) + A(1,2) & 7 \\
B(2,2) &= A(2,1) + B(1,2) & 8
\end{align*}
\]
Assume 4 stage pipeline

<table>
<thead>
<tr>
<th>Time (n cycles)</th>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Stage 3</th>
<th>Stage 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total time = 15 cycles.
Bad pipeline utilization.
After unroll and jam

do \forall j = 1, m, 2
  do i = 1, n
    \begin{align*}
    A(i, j) &= A(i+1, j) + A(i-1, j) \\
    B(i, j) &= A(i, j-1) + B(i-1, j)
    \end{align*}
  enddo
enddo

do i = 1, n
  \begin{align*}
  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)
  \end{align*}
enddo
endo

↓ "Jam"

do j = 1, m, 2
  do i = 1, n
    \begin{align*}
    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)
    \end{align*}
  enddo
endo
\[
\begin{align*}
A(1,1) &= A(2,1) + A(0,1) & \text{1} \\
B(1,1) &= A(1,0) + B(0,1) & \text{2} \\
A(1,2) &= A(2,2) + A(0,2) & \text{5} \\
B(1,2) &= A(1,1) + B(0,2) & \text{6} \\
A(2,1) &= A(3,1) + A(1,1) & \text{3} \\
B(2,1) &= A(2,0) + B(1,1) & \text{4} \\
A(2,2) &= A(3,2) + A(1,2) & \text{7} \\
B(2,2) &= A(3,1) + B(1,2) & \text{8}
\end{align*}
\]

**pipeline stages**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>8</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

**Total time = 13 cycles**

→ Better pipeline utilization
An example of iterative refinement

Program $P_0$

\[
\begin{align*}
X_0 & : \quad \text{for } i \\
& \quad \text{for } j \\
& \quad X^2(i,j) = A(i,j) - A(i-1,j) \\
& \quad \text{endfor} \\
& \text{endfor} \\
Y_0 & : \quad \text{for } j \\
& \quad \text{for } i \\
& \quad Y^2(i,j) = B(i,j) - B(i,j-1) \\
& \quad \text{endfor} \\
& \text{endfor} \\
\end{align*}
\]

Target Machine: MIMD hypercube

$\Delta(x'_0(i)) = i^{\text{th}} \text{ column}$

$\Delta(y'_0(j)) = j^{\text{th}} \text{ row}$

\[\text{dependence satisfied by communication} \]

\[\text{dependence satisfied locally} \]
Where does the user fit in?

Each level of the map problem \( \rightarrow \) hardware

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

User may find "easily" as parallelized at Caltech

Numerical Procedure

add information

loss of FORTRAN information

Hardware parallelization

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