

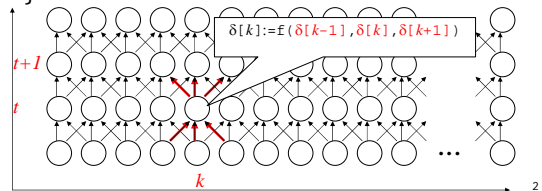
Data Distributions

Distribution Functions
Distributions with Redundancies
Redistribution as Configuration Problem

Example

```

1. for (t=1; t<T; t++) {
2.   forall (k=1; k<D; k++) in parallel {
3.      $\delta^{t+1}[k] = f(\delta^t[k-1], \delta^t[k], \delta^t[k+1])$ 
4.   }
5. }
    
```



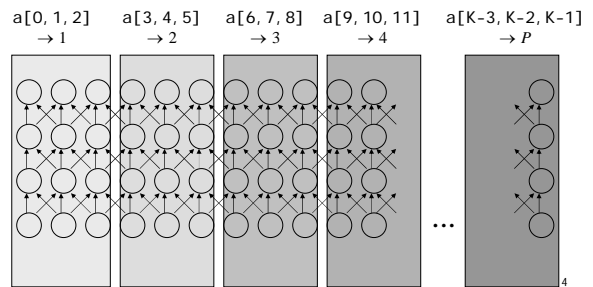
Mapping to distributed systems

- Two principle approaches:
 - Distribute the data
 - Each array element $\delta[k]$ is assigned to a processor
 - Computation follows:
 - All computations defining (writing) $\delta^{t+1}[k]=...$, $t=1..T$ are executed on the processor $\delta[k]$ is assigned to
 - Owner computes rule
 - Schedule the computations
 - Each computation is scheduled individually
 - Consumed array elements $\delta[k-1]$ $\delta[k]$ $\delta[k+1]$ and produced array elements $\delta^{t+1}[k]$ become local variables of the tasks

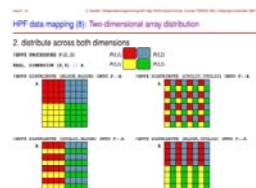
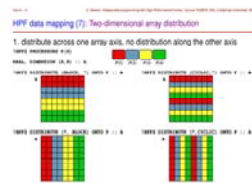
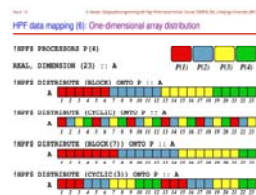
3

Distribute the Data

Distribution d is a function: $a \rightarrow \{1...P\}$



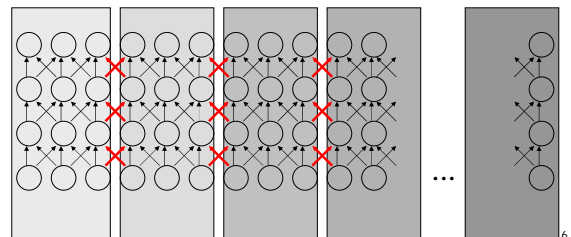
4



Blockwise:
 $d(a[i]) = i \div n/P$
Cyclic:
 $d(a[i]) = i \bmod P$
Block-cyclic:
 $d(a[i]) = (i \div \text{div } c) \bmod P$
Generalized to more dimensions

Good Data Distribution

- Compare execution costs in cost model
- In general: compute locally, avoid communications and synchronizations



6

Problem I: Alignment

Example Vector Product:

$$y = a \bullet x$$

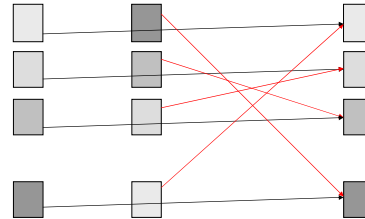
$$y_i = a_i x_i$$

1. for all (i=0; i<n; i++) in parallel {
2. y[i]=a[i]*x[i]
3. }
4. }

7

Align distribution of arrays

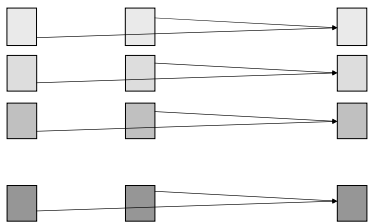
Vector a distributed blockwise
 Vector x distributed blockwise, **not** aligned to a
 Vector y then computed **with** communication



8

Align distribution of arrays

Vector a distributed blockwise
 Vector x distributed blockwise, aligned to a
 Vector y (aligned to a, x) then computed **without** any communication



9

Problem II: Redundancy

Example: Matrix-Vector Multiplication:

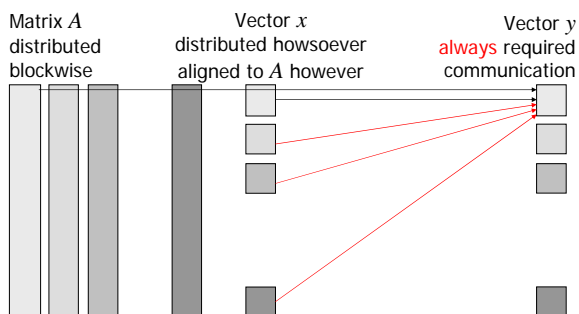
$$y = A x$$

$$y_i = a_{i,1} x_1 + \dots + a_{i,n} x_n$$

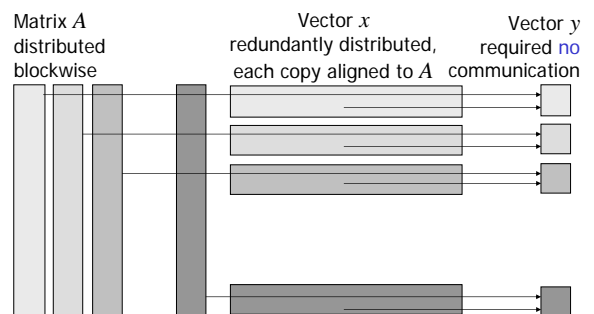
1. for all (i=0; i<n; i++) in parallel {
2. y[i]=0;
3. for (j=0; j<n; j++) {
4. y[i]=y[i] + a[i,j]*x[j]
5. }
6. }

10

Distribution function for arrays



Redundant Distribution



Redundancy

- Cannot be expressed by a function
 - Relation between array elements and processors
 - If derived automatically, larger solution space
- Could save communication
 - Costs local computation time, e.g., due to caching effects on processors
 - Reduced communication costs vs. increased memory access costs can only be biased by systematically in a more elaborated cost model including memory hierarchies
 - Too complicated to automatically optimize for

13

Problem III: Changing Alignment

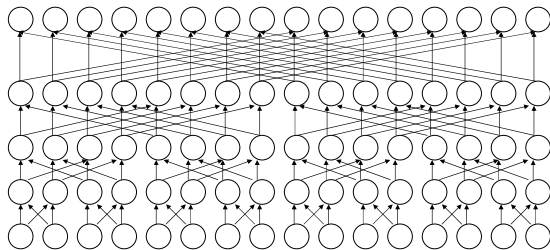
- Example: FFT (with $\omega=n$ -th unity root)

```

1. foral l (i=0; i<n; i++) in parallel
2.   x[i]=x[r(i)];
3.   for (i=0; i<log(n); i++) {
4.     foral l (j=0; j<n; j++) in parallel {
5.       if (j mod pow(2, i) < pow(2, i-1))
6.         x[j]=x[j] +  $\omega^{j_{2^i}}$  * x[j+pow(2, i-1)];
7.       el se
8.         x[j]=x[j -pow(2, i-1)] +  $\omega^{j_{2^i}}$  * x[j]
9.     }
10.  }
```

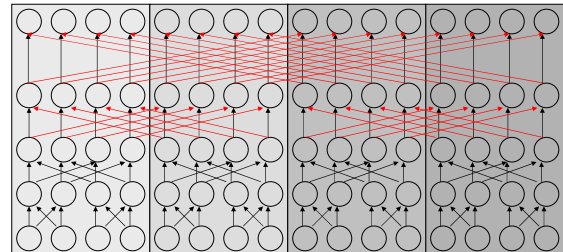
14

FFT Dependency Graph $n=16$



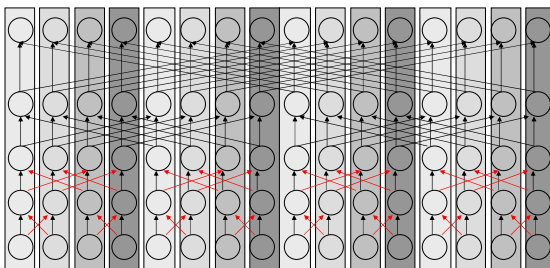
15

Block distribution $P=4$



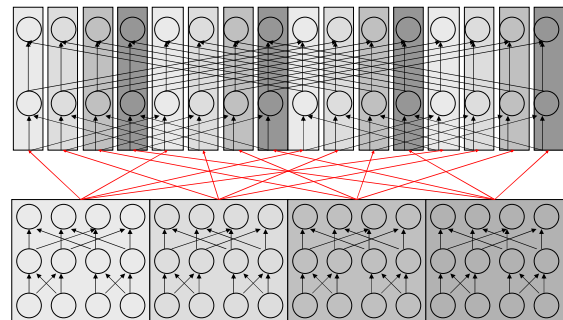
16

Cyclic distribution $P=4$



17

Redistribution (only one 4-relation)



18

Redistributions

- Cannot be expressed by relation between array elements and processors
 - Requires relation between array elements, iteration vectors (time axis) and processors
 - If derived automatically, larger solution space
- Could save communication
 - Sometimes it only bundles communication
 - Could only be biased in a cost model including communication parameters as functions on the message size

19

Problem IV: Composition

- Example: Polynomial Multiplication:

$$p(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1},$$

$$q(x) = b_0 + b_1x + \dots + b_{m-1}x^{m-1}$$

$$p(x)q(x) = c_0 + c_1x + \dots + c_{n+m-2}x^{n+m-1},$$

$$c_k = \sum_{j \in [0, k]} a_j b_{k-j}, \quad k \in [0, n+m-2]$$

- Computation by:

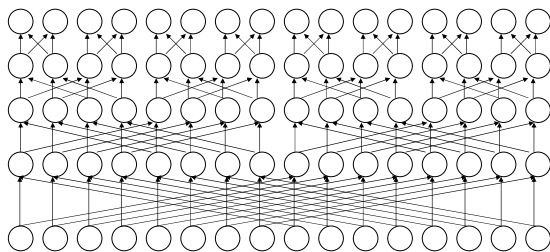
$$c = FFT^{-1}(FFT(a') \bullet FFT(b'))$$

$$a'_i = a_j, \quad i \in [0, n-1] \quad b'_i = b_j, \quad i \in [0, m-1]$$

$$a'_i = 0, \quad i \in [n, n+m-2] \quad b'_i = 0, \quad i \in [m, n+m-2]$$

20

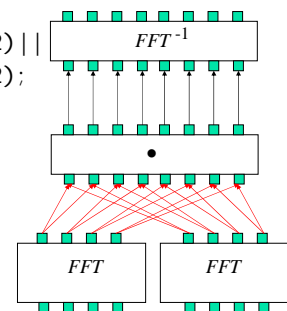
FFT⁻¹ Dependency Graph $n=16$



21

Implementation A

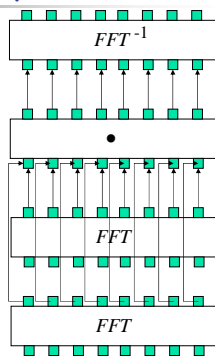
- parado
- $a' = FFT(a', P/2) \parallel$
- $b' = FFT(b', P/2);$
- $c' = \bullet(a', b', P);$
- $c = FFT^{-1}(c', P)$



22

Implementation (B)

- $a' = FFT(a, P);$
- $b' = FFT(b, P);$
- $c' = \bullet(a', b', P);$
- $c = FFT^{-1}(c', P)$



23

Composition

- Composition
 - Sequential P, P' : output distribution of P must conform to input distribution of P'
 - Parallel $P \parallel P'$: distribution of P uses p processors, distribution of P' uses p' processors where $p+p'=P$
- Optimal local solutions could be suboptimal globally
- Solutions:
 - Perform redistribution
 - Find globally good distribution

24

Conclusion of Problems

- Find appropriate distributions and alignments
- Introduce redundancy if applicable
- Redistribute within loops
- Compose according to constraints – redistribute/find globally good distributions

25

Approach

- Propose distribution functions d for each individual parallel assignment
- For each pair of distribution functions (d, d') for sequences of parallel assignments $S; S'$
 - Extend d such that (d, d') does not require communication (introduce redundancy)
 - Calculate redistribution costs for (d, d')
- Find the global optimum configuration

26

1. Propose distribution functions

- Refers to distribution of computed array d_{out}
- Induces a distribution and alignment of the input array(s) d_{in} (eventually redundant)
- Arbitrary basis for proposals
 - Analyze dependencies
 - Analyze task graph
 - Propose usual suspects (e.g. block, cyclic, ...)
 - Proposals of programmer (e.g. HPF directives)

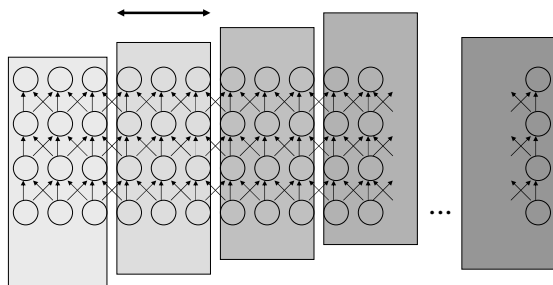
27

Example

- for ($t=1; t<T; t++$) {
 - for all ($k=1; i<D; k++$) in parallel {
 - $\delta^{t+1}[k] = f(\delta^t[k-1], \delta^t[k], \delta^t[k+1])$
 - }
 - }
- Assign: $\delta^{t+1}[k] = f(\delta^t[k-1], \delta^t[k], \delta^t[k+1])$
 - Proposal block distribution:
 - $d_{out}(\delta[k]) = k \text{ div } n/P$
 - $d_{in}(\delta[k]) = k \text{ div } n/P,$
 - $d_{in}(\delta[k-1]) = k \text{ div } n/P$
 - $d_{in}(\delta[k+1]) = k \text{ div } n/P$
 - d_{in} is redundant

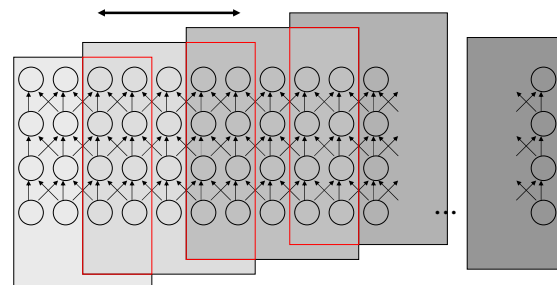
28

d_{out}



29

d_{in}



30

2. Sequences of parallel assignments

- Not uniquely defined at compile time
- Treat loops and branches conservatively
- No redundancy/redistribution costs for (d_{out}, d'_{in}) iff $d'_{in} \leq d_{out}$
 - Note that d_{out}, d'_{in} are relations, i.e. sets of pairs
 - \leq defined to be partial subset order relation \subseteq
- Several iterations possible

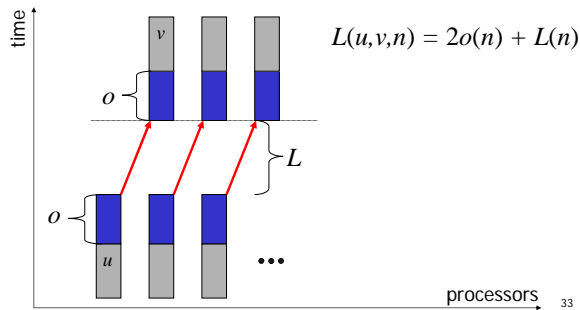
31

Example

- for (t=1; t<T; t++) {
 - foral I (k=1; i<D; k++) in parallel {
 - $\delta^{t+1}[k] = f(\delta^t[k-1], \delta^t[k], \delta^t[k+1])$
 - }
 - }
- New redundant distribution $d'_{out} = d_{in}$ because of loop, compute d'_{in} accordingly
 - Communication cost $L(d'_{out}, d_{in}) = 0$ by definition
 - Redistribution costs $L(d_{out}, d_{in}), L(d_{out}, d'_{in}), L(d'_{out}, d'_{in})$ depend on cost model

32

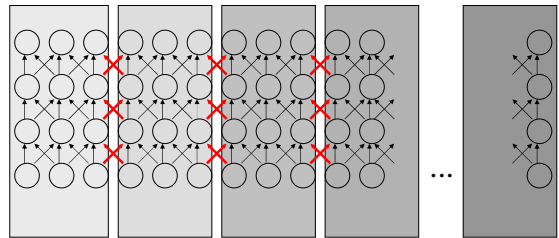
LogP Communication Step



Example with LogP as cost model

$$L(d_{out}, d_{in}) = L(d'_{out}, d'_{in}) = 2(2o(2) + L(2)) \quad (2 \text{ elements overlap})$$

$$L(d_{out}, d'_{in}) = 2(2o(4) + L(4)) \quad (4 \text{ elements overlap})$$



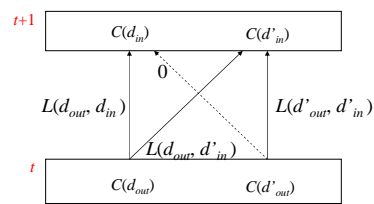
3. Global optimum configuration

- Idea for not iteratively composed programs:
 - Compute the basic block graph of the program
 - Assign computation costs for each $C(d_{out})$ to nodes (this may be different since redundant distributions require redundant computations)
 - Assign Redistribution costs $L(d_{out}, d'_{in})$ to the edges
 - Find minimum path in graph
- NP hard problem in general (polynomial for goto free languages, ok in practice)

35

Example (unrolled assignments)

- foral I (k=1; i<D; k++) in parallel
- $\delta^{t+1}[k] = f(\delta^t[k-1], \delta^t[k], \delta^t[k+1])$
- foral I (k=1; i<D; k++) in parallel
- $\delta^{t+2}[k] = f(\delta^{t+1}[k-1], \delta^{t+1}[k], \delta^{t+1}[k+1])$



3. Iterations

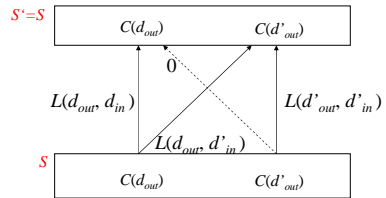
- Brute force (exact)
 - Unroll loop
 - Treat as them before
 - Unavoidable, if costs depend on the iteration
- Approximation
 - Let S, S' be last and first loop statement, resp.
 - Assume an artificial redistribution $d_{out}(S), d_{in}(S')$ also for the last iteration
 - Let S'' be the first statement after loop
 - Connect $d_{out}(S'), d_{in}(S'')$

37

Example

```

1. for (t=1; t<T; t++) {
2.   forall (k=1; i<D; k++) in parallel {
3.      $\delta^{i+1}[k] = f(\delta^i[k-1], \delta^i[k], \delta^i[k+1])$ 
4.   }
5. }
  
```



38

Conclusion

- Only approximations of the optimum
 - Compilers can give good results
 - Sometimes not good enough
 - Therefore programmers must be able to find better solutions for specific problems "by hand" (preferably "by head")
- Alternative to data distribution: task scheduling

39