Dynamic Programming and MPI

Mark Greenstreet

CpSc 418 – Oct. 30, 2012

Lecture Outline

- Dynamic Programming
  - The editing distance problem.
  - Computing editing distance with dynamic programming.
  - Parallel Implementation
- Implementing Dynamic Programming in MPI

Genome Comparison

- Poodles and German Shepherds both descended from wolves?
- Which is the closer descendant?
- Let P, G and W be the genomes (strings) for a poodle, a german shepherd, and a wolf.
  - Compute a "distance" from W to P and from W to G.
  - How?
  - Consider editing operations to transform W to P (or vice-versa):
    - insert a character, c_1 into the W string;
    - delete a character, c_2 from the W string;
    - replace a character, c_3, in the W string with a new character, c_4.
  - Assign a cost to each of these operations according to how likely the mutation is.
  - Find the minimum cost sequence of edits that transforms W to P.
  - The cost of this sequence of edits is the editing distance between W and P, $\mathrm{edist}(W, P)$.

Example

What is the editing distance between `"hello world"` and `"hew gold"`?

- Exploring all possible sequences of edits would be very expensive (i.e. exponential cost).
- Key idea: what if we knew the optimal editing sequences for
  - `"hello world"` $\rightarrow$ `"hew gol"`,
  - `"hello worl"` $\rightarrow$ `"hew gold"`, and
  - `"hello worl"` $\rightarrow$ `"hew gol"`,

  then, edist(`"hello world"`, `"hew gold"`) would be

$$\min\left(\begin{array}{l} \text{edist}(\texttt{"hello world", "hew gol"}) + \textit{cost}(\textit{insert}\texttt{'d'}), \\ \text{edist}(\texttt{"hello worl", "hew gold"}) + \textit{cost}(\textit{delete}\texttt{'d'}), \\ \text{edist}(\texttt{"hello worl", "hew gol"}) \ + \mathit{0} \end{array} \right)$$

Building a cost tableau

- Let prefix(n, s) be the first n characters of string s.
- We'll construct an array, `cost[i,j]` with
  `cost[i,j] = edist(prefix(i, "hello world"), prefix(j, "hew gold"))`.
- Let
  - $p_{ins} = p_{del}$ = cost of inserting or deleting a character.
  - p_{rpl} = of replacing a character.
- When `i` and `j` are both greater than 1:

$$
\texttt{cost[i,j]} = \min(\begin{array}{l} \texttt{cost[i-1,j]} + p_{del}, \\ \texttt{cost[i,j-1]} + p_{ins}, \\ \texttt{cost[i-1,j-1]} + p_{rpl} \end{array}).
$$

Getting Started

- `cost[0,0]` $=$ 0: the empty-string matches the empty-string.
- `cost[i,0]` $=$ `i` $* p_{del}$:
  - We can't quite use the rule from the previous slide, because we don't have `cost[i,-1]` or `cost[i-1,j-1]`.
  - `cost[i,0]` is the editing distance from a string with `i` characters to the empty string.
  - The only way to transform a string with `i` characters to the empty string is to delete all the characters.
  - $\therefore$ `cost[i,0]` $=$ `i` $* p_{del}$.
- `cost[0,j]` $=$ `j` $* p_{ins}$:
  In this case, we're inserting `j` characters to transform the empty string into a string with `j` characters.

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	$\mathtt{j} = 0$	$\mathtt{'h'}$ $\mathtt{j} = 1$	$\mathtt{'e'}$ $\mathtt{j} = 2$	$\mathtt{'w'}$ $\mathtt{j} = 3$	$\mathtt{'\ '}$ $\mathtt{j} = 4$	$\cdots$ $\cdots$
$\mathtt{i} = 0$						
$\mathtt{'h'},\ \mathtt{i} = 1$						
$\mathtt{'e'},\ \mathtt{i} = 2$						
$\mathtt{'l'},\ \mathtt{i} = 3$						
$\mathtt{'l'},\ \mathtt{i} = 4$						
$\mathtt{'o'},\ \mathtt{i} = 5$						
$\vdots$						

first overlay final tableau

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	$j=0$	'h' $j=1$	'e' $j=2$	'w' $j=3$	' ' $j=4$	... ...
$i=0$	0					
'h', $i=1$						
'e', $i=2$						
'l', $i=3$						
'l', $i=4$						
'o', $i=5$						
⋮						

first overlay final tableau

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	$j = 0$	'h' $j = 1$	'e' $j = 2$	'w' $j = 3$	' ' $j = 4$	$\dots$ $\dots$
$i = 0$	0	2				
'h', $i = 1$						
'e', $i = 2$						
'l', $i = 3$						
'l', $i = 4$						
'o', $i = 5$						
$\vdots$						

first overlay final tableau

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	$j = 0$	'h' $j = 1$	'e' $j = 2$	'w' $j = 3$	' ' $j = 4$	... ...
$i = 0$	0	2	4			
'h', $i = 1$						
'e', $i = 2$						
'l', $i = 3$						
'l', $i = 4$						
'o', $i = 5$						
⋮						

first overlay final tableau

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	$j = 0$	'h' $j = 1$	'e' $j = 2$	'w' $j = 3$	' ' $j = 4$	$\cdots$
$i = 0$	0	2	4	6	8	$\cdots$
'h', $i = 1$						
'e', $i = 2$						
'l', $i = 3$						
'l', $i = 4$						
'o', $i = 5$						
$\vdots$						

first overlay final tableau

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	$\texttt{j} = 0$	'h' $\texttt{j} = 1$	'e' $\texttt{j} = 2$	'w' $\texttt{j} = 3$	' ' $\texttt{j} = 4$	$\cdots$ $\cdots$
$\texttt{i} = 0$	0	2	4	6	8	$\cdots$
'h', $\texttt{i} = 1$	2					
'e', $\texttt{i} = 2$						
'l', $\texttt{i} = 3$						
'l', $\texttt{i} = 4$						
'o', $\texttt{i} = 5$						
$\vdots$						

first overlay final tableau

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	$\texttt{j} = 0$	$\texttt{'h'}$ $\texttt{j} = 1$	$\texttt{'e'}$ $\texttt{j} = 2$	$\texttt{'w'}$ $\texttt{j} = 3$	$\texttt{' '}$ $\texttt{j} = 4$	$\dots$ $\dots$
$\texttt{i} = 0$	0	2	4	6	8	$\dots$
$\texttt{'h'},\ \texttt{i} = 1$	2					
$\texttt{'e'},\ \texttt{i} = 2$	4					
$\texttt{'l'},\ \texttt{i} = 3$	6					
$\texttt{'l'},\ \texttt{i} = 4$	8					
$\texttt{'o'},\ \texttt{i} = 5$	10					
$\vdots$	$\vdots$					

first overlay final tableau

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	$\texttt{j} = 0$	'h' $\texttt{j} = 1$	'e' $\texttt{j} = 2$	'w' $\texttt{j} = 3$	' ' $\texttt{j} = 4$	$\cdots$ $\ldots$
$\texttt{i} = 0$	0	2	4	6	8	$\cdots$
'h', $\texttt{i} = 1$	2	0				
'e', $\texttt{i} = 2$	4					
'l', $\texttt{i} = 3$	6					
'l', $\texttt{i} = 4$	8					
'o', $\texttt{i} = 5$	10					
$\vdots$	$\vdots$					

first overlay final tableau

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	$\texttt{j} = 0$	$\texttt{'h'}$ $\texttt{j} = 1$	$\texttt{'e'}$ $\texttt{j} = 2$	$\texttt{'w'}$ $\texttt{j} = 3$	$\texttt{' '}$ $\texttt{j} = 4$	$\cdots$ $\cdots$
$\texttt{i} = 0$	0	2	4	6	8	$\cdots$
$\texttt{'h'},\ \texttt{i} = 1$	2	0	2			
$\texttt{'e'},\ \texttt{i} = 2$	4					
$\texttt{'l'},\ \texttt{i} = 3$	6					
$\texttt{'l'},\ \texttt{i} = 4$	8					
$\texttt{'o'},\ \texttt{i} = 5$	10					
$\vdots$	$\vdots$					

first overlay final tableau

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	$j = 0$	'h' $j = 1$	'e' $j = 2$	'w' $j = 3$	' ' $j = 4$	$\cdots$ $\ldots$
$i = 0$	0	2	4	6	8	$\cdots$
'h', $i = 1$	2	0	2	4	6	$\cdots$
'e', $i = 2$	4					
'l', $i = 3$	6					
'l', $i = 4$	8					
'o', $i = 5$	10					
$\vdots$	$\vdots$					

first overlay final tableau

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	$j = 0$	'h' $j = 1$	'e' $j = 2$	'w' $j = 3$	' ' $j = 4$	… …
$i = 0$	0	2	4	6	8	…
'h', $i = 1$	2	0	2	4	6	…
'e', $i = 2$	4	2				
'l', $i = 3$	6					
'l', $i = 4$	8					
'o', $i = 5$	10					
⋮	⋮					

first overlay final tableau

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	$j = 0$	'h' $j = 1$	'e' $j = 2$	'w' $j = 3$	' ' $j = 4$	… …
$i = 0$	0	2	4	6	8	…
'h', $i = 1$	2	0	2	4	6	…
'e', $i = 2$	4	2	0			
'l', $i = 3$	6					
'l', $i = 4$	8					
'o', $i = 5$	10					
⋮	⋮					

first overlay final tableau

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	$j = 0$	'h' $j = 1$	'e' $j = 2$	'w' $j = 3$	' ' $j = 4$	$\cdots$ $\ldots$
$i = 0$	0	2	4	6	8	$\cdots$
'h', $i = 1$	2	0	2	4	6	$\cdots$
'e', $i = 2$	4	2	0	2	4	$\cdots$
'l', $i = 3$	6					
'l', $i = 4$	8					
'o', $i = 5$	10					
$\vdots$	$\vdots$					

first overlay final tableau

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	$\mathtt{j} = 0$	$\mathtt{'h'}$ $\mathtt{j} = 1$	$\mathtt{'e'}$ $\mathtt{j} = 2$	$\mathtt{'w'}$ $\mathtt{j} = 3$	$\mathtt{'\ '}$ $\mathtt{j} = 4$	$\cdots$ $\ldots$
$\mathtt{i} = 0$	0	2	4	6	8	$\cdots$
$\mathtt{'h'},\ \mathtt{i} = 1$	2	0	2	4	6	$\cdots$
$\mathtt{'e'},\ \mathtt{i} = 2$	4	2	0	2	4	$\cdots$
$\mathtt{'l'},\ \mathtt{i} = 3$	6	4				
$\mathtt{'l'},\ \mathtt{i} = 4$	8					
$\mathtt{'o'},\ \mathtt{i} = 5$	10					
$\vdots$	$\vdots$					

first overlay final tableau

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	$\texttt{j} = 0$	$\texttt{'h'}$ $\texttt{j} = 1$	$\texttt{'e'}$ $\texttt{j} = 2$	$\texttt{'w'}$ $\texttt{j} = 3$	$\texttt{' '}$ $\texttt{j} = 4$	... ...
$\texttt{i} = 0$	0	2	4	6	8	...
$\texttt{'h'}$, $\texttt{i} = 1$	2	0	2	4	6	...
$\texttt{'e'}$, $\texttt{i} = 2$	4	2	0	2	4	...
$\texttt{'l'}$, $\texttt{i} = 3$	6	4	2			
$\texttt{'l'}$, $\texttt{i} = 4$	8					
$\texttt{'o'}$, $\texttt{i} = 5$	10					
⋮	⋮					

first overlay final tableau

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	$j = 0$	'h' $j = 1$	'e' $j = 2$	'w' $j = 3$	' ' $j = 4$	$\cdots$ $\cdots$
$i = 0$	0	2	4	6	8	$\cdots$
'h', $i = 1$	2	0	2	4	6	$\cdots$
'e', $i = 2$	4	2	0	2	4	$\cdots$
'l', $i = 3$	6	4	2	3		
'l', $i = 4$	8					
'o', $i = 5$	10					
$\vdots$	$\vdots$					

first overlay final tableau

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	$\texttt{j} = 0$	$\texttt{'h'}$ $\texttt{j} = 1$	$\texttt{'e'}$ $\texttt{j} = 2$	$\texttt{'w'}$ $\texttt{j} = 3$	$\texttt{' '}$ $\texttt{j} = 4$	$\cdots$ $\ldots$
$\texttt{i} = 0$	0	2	4	6	8	$\cdots$
$\texttt{'h'}, \ \texttt{i} = 1$	2	0	2	4	6	$\cdots$
$\texttt{'e'}, \ \texttt{i} = 2$	4	2	0	2	4	$\cdots$
$\texttt{'l'}, \ \texttt{i} = 3$	6	4	2	3	5	$\cdots$
$\texttt{'l'}, \ \texttt{i} = 4$	8					
$\texttt{'o'}, \ \texttt{i} = 5$	10					
$\vdots$	$\vdots$					

first overlay final tableau

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	'h'	'e'	'w'	' '	…
	j = 0	j = 1	j = 2	j = 3	j = 4	…
i = 0	0	2	4	6	8	…
'h', i = 1	2	0	2	4	6	…
'e', i = 2	4	2	0	2	4	…
'l', i = 3	6	4	2	3	5	…
'l', i = 4	8	6	4	5	6	…
'o', i = 5	10					
⋮	⋮					

first overlay final tableau

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	'h'	'e'	'w'	' '	$\cdots$
	j = 0	j = 1	j = 2	j = 3	j = 4	$\cdots$

		'h'	'e'	'w'	' '	$\cdots$
	j = 0	j = 1	j = 2	j = 3	j = 4	$\cdots$
i = 0	0	2	4	6	8	$\cdots$
'h', i = 1	2	0	2	4	6	$\cdots$
'e', i = 2	4	2	0	2	4	$\cdots$
'l', i = 3	6	4	2	3	5	$\cdots$
'l', i = 4	8	6	4	5	6	$\cdots$
'o', i = 5	10	8	6	7	8	$\cdots$
$\vdots$	$\vdots$					

first overlay final tableau

Let's do it

- Assume $p_{ins} = p_{del} = 2$, $p_{rpl} = 3$.
- The tableau:

	'h'	'e'	'w'	' '	$\cdots$
	j = 0	j = 1	j = 2	j = 3	j = 4	$\ldots$
i = 0	0	2	4	6	8	$\cdots$
'h', i = 1	2	0	2	4	6	$\cdots$
'e', i = 2	4	2	0	2	4	$\cdots$
'l', i = 3	6	4	2	3	5	$\cdots$
'l', i = 4	8	6	4	5	6	$\cdots$
'o', i = 5	10	8	6	7	8	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

first overlay final tableau

The final tableau

		'h'	'e'	'w'	' '	'g'	'o'	'l'	'd'
	0	2	4	6	8	10	12	14	16
'h'	2	0	2	4	6	8	10	12	14
'e'	4	2	0	2	4	6	8	10	12
'l'	6	4	2	3	5	7	9	8	10
'l'	8	6	4	5	6	8	10	9	11
'o'	10	8	6	7	8	9	8	10	12
' '	12	10	8	9	7	9	10	11	13
'w'	14	12	10	8	9	10	12	13	14
'o'	16	14	12	10	11	12	10	12	14
'r'	18	16	14	12	13	14	12	13	15
'l'	20	18	16	14	15	16	14	12	14
'd'	22	20	18	16	17	18	16	14	12

Observations

- We can compute the editing distance between two strings of length N in $O(N^2)$ sequential time.
  - A single tableau entry can be computed in $O(1)$ time.
  - There are $O(N^2)$ tableau entries.
- The algorithm can also provide a sequence of editing operation that achieves the minimum cost.
  - After computing the tableau, work backwards from the lower-right corner to the upper left.
  - This takes $O(N)$ additional time.
  - Warning: it also requires $O(N^2)$ storage.
    - ★ This may be impractical for larger problems.
  - We can do better, but that's not the topic of this course. ☺
- If we don't need the sequence of editing operations, $O(N)$ space is sufficient.
  - Only need to store row $i-1$ until we're done computing row i.

Implementing the code

Code sketch:

```
int edist(char *top, char *left, Penalty *p) {
    ...
    for each row i { // each char of top
        for each column j{ // each char of left
            compute entry tableau[i,j] based on entries
            tableau[i-1,j-1], tableau[i-1,j], and tableau[i,j-1].
        }
    }
    return(tableau(nrow-1, ncols-1));
}
```

Warning: storing the entire `tableau` array would require $O(N^2)$ space (as noted on slide 9).

$O(N)$ Storage

- Use an array, `cost[0..(N-1)]`. Initially, `cost[j] = 2*j`.
- The "`for j`" loop from slide 10 will maintain `cost` such that when the loop condition is tested:
  - All elements of cost with indices less than `j` have values for the current row (i.e. row `i`).
  - All elements of cost with indices greater than or equal to `j` have values for the previous row (i.e. row `i-1`).
- One tricky point: computing `cost[j]` (i.e. *tableau[i,j]*) requires the value of *tableau[i-1,j-1]*, but we've already set `cost[j-1]` to the value of *tableau[i,j-1]*.
  - Solution. Use local variables `cost_n` and `cost_nw`:
    - ★ `cost_n` is the cost of the tableau entry to the "north" of the entry currently being computed; i.e., `cost_n =` *tableau[i-1,j]*.
    - ★ `cost_nw` is the cost of the tableau entry to the "northwest" of the entry currently being computed; i.e., `cost_nw =` *tableau[i-1,j-1]*.
  - At the beginning of the body of the `for j` loop:
    - ★ Set `cost_nw` to `cost_n`.
    - ★ Set `cost_n` to `cost[j]`.
      Note that `cost[j]` hasn't been updated yet; so it still has the value of *tableau[i-1,j]*.

Editting Distance In C

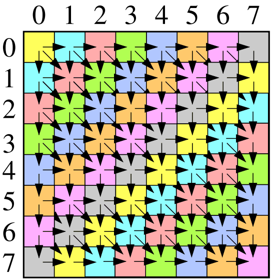
```
int edist(char *top, char *left, Penalty *p) {
   int ncols = strlen(top);
   int nrows = strlen(left);
   int *cost = (int *)malloc(ncols*sizeof(int));
   for(int j = 0; j < ncols; j++)
      cost[j] = 2*(j+1); // initialize cost
   for(int i = 0; i < nrows; i++) { // each tableau row
      int cost_n = 2*i;
      int cost_w = 2*(i+1);
      for(int j = 0; j < ncols; j++) { // each tableau column
         int cost_nw = cost_n;
         cost_n = cost[j];
         cost[j] = min(
            cost_nw + ((top[j]==left[i]) ? 0 : p->replace),
            min(cost_n, cost_w) + p->insdel);
         cost_w = cost[j];
   } } return(cost[ncols-1]);
}
```

Code at: simple_edist.c

Do it in parallel

- Find the parallelism
- Find the overhead
  - Commnication
  - Idle processis
- Implement the code (in MPI)
- Measure the performance

Dependencies



A tableau element can be updated when the values for its incoming arrows are available.

- Initially, *tableau[0,0]* can be computed.
- Second, *tableau[0,1]* and *tableau[1,0]* can be computed in parallel.
- Third, *tableau[0,2]*, *tableau[1,1]*, and *tableau[2,0]* can be computed in parallel.

First Parallel Version

In Peril-L (see Oct. 25 slides)

```
for i in 0..(2N-1) {
    forall j in 0..i {
        update tableau[j,i-j];
    }
}
```

- Each element update involves six communication actions:
  - Receive values from N, W, and NW neighbours.
  - Send values to S, E, and SE neighbours.
- Communication cost will dominate computation.
- This is an example of “unlimitted” parallelism leading to an inefficient algorithm.

Partition Work into Blocks

Divide the tableau into $B \times B$ blocks.

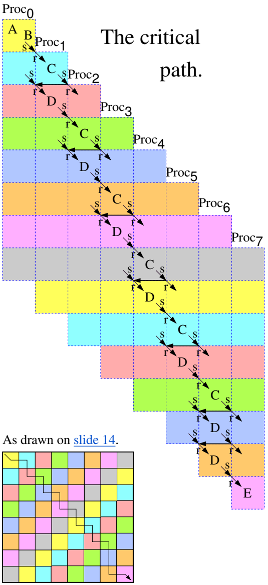
- Computing the tableau entries for a $B \times B$ block requires
  - $O(B^2)$ computation
  - 4 communications – the "diagonal" values just involve appending one more element to each vector sent.
  - Each communication operation transfers $B + 1$ values.
- Simple approach: compute editing distance between two strings of length N using P processors.
  - Divide tableau into P^2 blocks, each of size $(N/P) \times (N/P)$.
  - Each processor is responsible for one column.
    - ★ The processor computes the tableau for the block from top-to-bottom.
    - ★ To work on a block, processors $1 \ldots P - 1$ must first receive the cost-vector from the processor on its left.
    - ★ When a processor finishes a block, it sends the cost vector for its right eedge to the processor on its right.
  - Each communication operation transfers B values.

Second Parallel Version

```
for d in 0..(2P-2) { // each of the 2P − 1 diagonals
   forall b in 0..max(d+1, 2P-(d+1)){ // each block along
      for i in 0..((N/P)-1) {           // the diagonal
         for j2 in 0..((N/P)-1) {
            update tableau[(N/P)*(d-b) + i2, (N/P)*b + j2]
} } } }
```

- This algorithm suffers from idle processors.
  - Initially, only one processor is active.
  - After the first procesor finishes its first block, two procesors are active.
  - All processors are active only when computing the blocks on the anti-diagonal.
  - So, we'd expect a maximum speed-up of about $P/2$.
- I'll implement and analyse this version anyway, and leave the improvements for a homework problem.

Performance (1/2)



- The pieces of the critical path:
  - A is the initial computation of the upper left box of the tableau by processor Proc_0.
  - B is the time for processor Proc_0 to send a message (the cost vector for the right edge of the tableau block it just evaluated) to processor Proc_1.
  - C is the time for a processor to receive a message, compute a block, send a message. The critical path continues on the same processor.
  - D is the time for a processor to receive a a message, compute a block, and send a message. The critical path continues on the next processor to the right.
  - E is the time for the rightmost processor to receive a message and update the final block to obtain the final cost.

Performance (2/2)

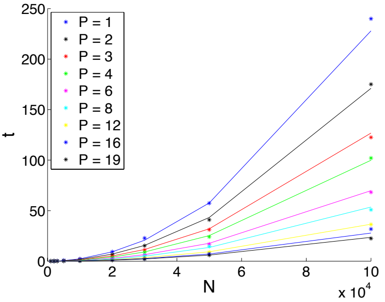
The total time:

- At each of the steps, a processor computes the tableau entries for a $(N/P) \times (N/P)$ block. There are $2P - 1$ such steps, for a total compute time of $t_{update}(2P - 1)N^2/P^2$ where t_{update} is the time to compute a single update of the tableau.
- At every step except for the last one, the processor sends a message to its successor. Likewise, at every step except for the first one, the processor receives a message from its predecessor. The total communication time is: $2(t_{send}(N/P) + t_{recv}(N/P))(P - 1)$, where $t_{send}(N/P)$ is the time to send a message of N/P cost values, and $t_{recv}(N/P)$ is the time to receive such a message.
- Assume that the time to send and receive a message with N/P elements is $t_0 + t_1(N/P)$, then the total time for the algorithm is:

$$t_{update}\frac{(2P-1)N^2}{P^2} + 2\left(t_0 + \frac{N}{P}t_1\right)(P-1)$$

For $N \gg P \gg 1$, this is approximately $2t_{update}N^2/P$, which means we expect a speed-up of roughly half the number of processors.

Let's try it



- I implemented the algorithm described above using MPI and ran it using the `lin01`... `lin25.ugrad.cs.ubc.ca` machines.
- Fitting the parameters of the model from the previous slide to the measured run-times yields:

$$t = \left(4.85 \cdot 10^{-3} + 7.82 \cdot 10^{-4} p + 1.77 \cdot 10^{-6} \frac{N}{P}(P-1) + 2.30 \cdot 10^{-8} \frac{N^2}{P^2}(2P-1)\right)$$

- This yields: $t_{update} \approx 23\mathrm{ns}$, $t_0 \approx 0.39\mathrm{ms}$, and $t_1 \approx 0.87\mu\mathrm{s}$.
  - The constant term, $4.85\mathrm{ms}$ didn't appear in the model on the previous slide. I included it to account for the fixed overheads in the algorithm, which apparently are fairly large.
  - The other terms are (surprisingly) reasonable ☺.

Full Disclosure

- To fit the model to the data, I discarded the data from the $P = 1$ case.
  - Visually, it was an outlier (too slow!).
  - My main focus is the parallel case anyway.
- Note that the t_{update} is dominant for large values of N, but the other parameters matter for small values of N.
  - For example, I don't want the "best-fit" for large N to produce a model that predicts negative run-times for small N.
  - So,
    - I did a least-squares (minimize the square of the absolute error) first to obtain an estimate of the parameters.
    - I fixed t_{update} to the value from that fit and re-fit the other parameters to minimize the square of the relative error.
    - I fixed the non t_{update} parameters and did one more least-sqares fit for t_{update} to minimize the square of the absolute error.

Announcements and reminders

Review

I'll add somthing for this.