

Bridging Models and Machines

PRAM – BSP – Delay Model – LogP

Problem

- What is the von-Neumann model to for parallel machines
 - Much more sensitive wrt. reflecting performance
 - Much more diverse in existing architectures
 - Message passing networks of different kinds
 - Internet
 - Shared and virtual shared memory machines
 - Vector machines ...
- Conflict between
 - Easy to program, portability of programs
 - Accurately reflecting performance

3

PRAM revisited

- + Easy to program
- + Portable programs
- Unrealistic assumptions like constant time memory access
- Expensive simulations on existing message architectures
 - Looks ok in the O -calculus
 - Large constants on message passing machines
 - But constant speed-up is all we can hope for

5

Machine Models

- What are models good for?
 - Abstracting from machine properties
 - Making programming simple
 - Making programs portable
 - Reflecting essential machine properties
 - Functionality (sure)
 - Costs (programmer should understand that a program is expensive when (s)he writes it) as long as it cannot be hidden by compilers
- Success of the von-Neumann machine model

2

Questions

- Evaluate the models with respect to
 - Programmability
 - Reality
 - Simulations
 - Compilations

4

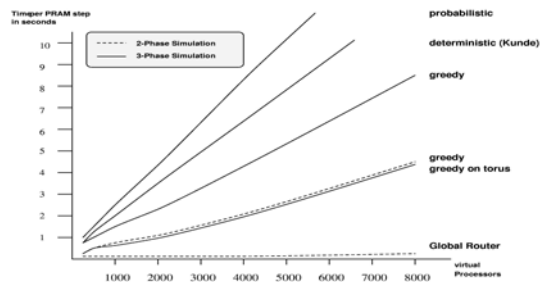
Theoretical simulation results

- Deterministic and probabilistic methods
- Deterministic
 - Each PRAM cell is stored on different nodes (memory organization scheme)
 - An general optimum memory organization scheme is unknown (only its existence for individual topologies)
 - E.g. $O(\log^2 p / \log \log p)$ for a p -PRAM step on a p -mesh
- Probabilistic
 - Probabilistic distribution of the memory cells
 - E.g. $O(\max(\log p, v / p))$ for a v -PRAM step on a p -CCC, p -hypercube, or p -butterfly (Valiant) - optimal if $v > p \log p$

6

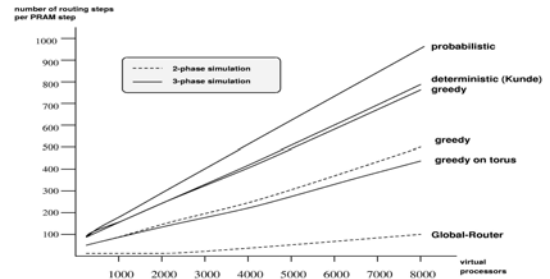
Measurements - time

(Zimmermann, Kumm) MASPAR MP-1 256 processors



Measurements - steps

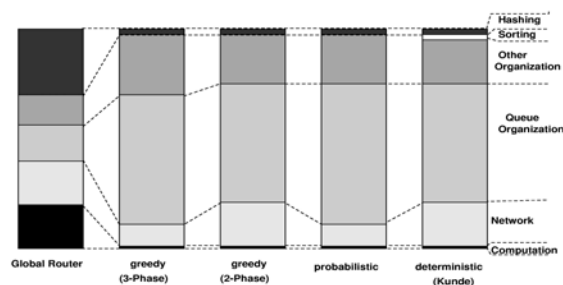
(Zimmermann, Kumm) MASPAR MP-1 256 processors



8

Measurements

MASPAR MP-1 256 processors

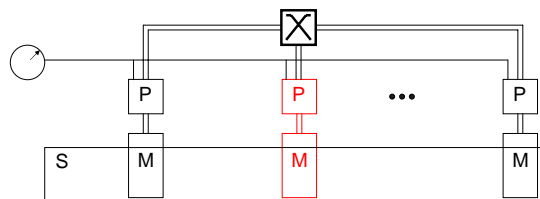


BSP (Valiant)

- Bulk-Synchronous Parallel Machine
- Avoids the costs in the PRAM simulation for hashing, sorting, queuing, other organizational tricks ☺
- Let the programmer handle this problem ☹
- Bridging model for parallel computation
 - Standard results on probabilistic PRAM simulations in Handbook of Theoretical CS are by Valiant
 - Even he obviously sees a need to get closer to reality

10

BSP (Valiant)



- Processor (P),
- Virtually shared (S) and/or local Memory (M),
- Common synchronization
- Router

11

BSP Computations

- In super-steps, each:
 - Processors read values required in a step
 - Perform computations locally
 - Store values computed in that step
 - Bulk-synchronize before the next step
- Periodicity of L for synchronization

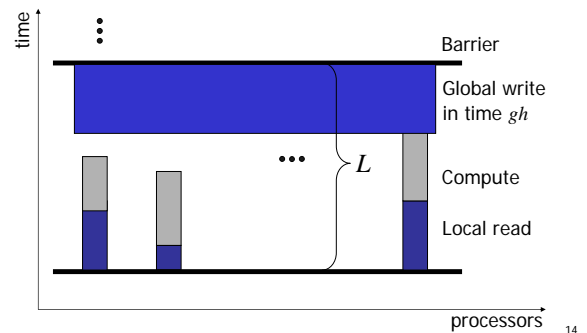
12

Cost Model

- Router can handle h -relations in time $hg' + s$
 - Number of messages sent or received h
 - Router throughput g'
 - Startup time s
 - For simplicity, define a g such that the router can handle h -relations in time hg for $h > h_0$ (some initial value) – e.g. take $g=2g'$ assuming $hg' > s$
- Router implementation is hidden in a library

13

Super-steps



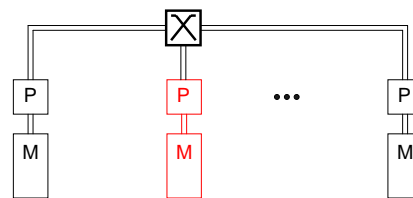
14

Periodic synchronization

- Assumed to be implemented in hardware
 - At least independent of the processors
 - Otherwise there wouldn't be any processor capacity left for computation in the super-steps
- Bound from below by the hardware
- Bound from above by the application
 - Larger super-steps means longer independent parallel computations without the need of establishing a consistent memory state
 - Requires higher granularity in the problem to allow that

15

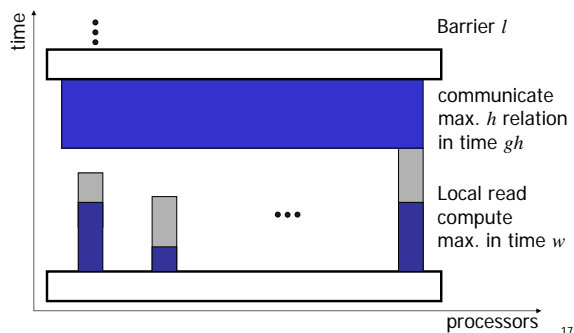
BSP (McColl)



- Processor (P),
- Memory (M),
- Common synchronization in software
- Network connected

16

Super-steps



17

Discussion Valiant vs. McColl

- McColl
 - Gives up periodicity L as unnecessary constraint
 - Introduces explicit synchronization time l accounting for synchronization in processors, i.e., sharing the hardware with computation and communication
 - Assumes message passing to address usual hardware
- Valiant
 - Preserves the ability of managed data distribution from the deterministic PRAM simulation
 - Allowing user defined data distribution if applicable

18

Design a BSP program

- Execution time:

$$T = \sum_{s \in \text{super-steps}} (\max_{i \in \text{procs}} w_{i,s} + \max_{i \in \text{procs}} h_{i,s} g + l)$$

- Implications for algorithm design:
 - Balance computation because of $\max_{i \in \text{procs}} w_i$
 - Balance communications because of $\max_{i \in \text{procs}} h_i g$
 - Minimize the number of super-steps because of $|\text{super-steps}| \times l$

19

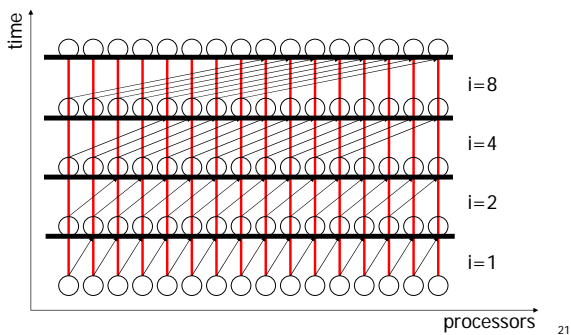
Example Prefix Sums – Plan A

```

1. for (p=0; p<n; p++) in parallel {
2.   right=init(p); left=0;
3.   target processor      value from local variable      to target
4.   processor             variable                       processor
5.   put(p+i, right, left);
6.   barrier_synchronize();
7.   if (p >= i)
8.     right=right+left;
9. }
10. }
```

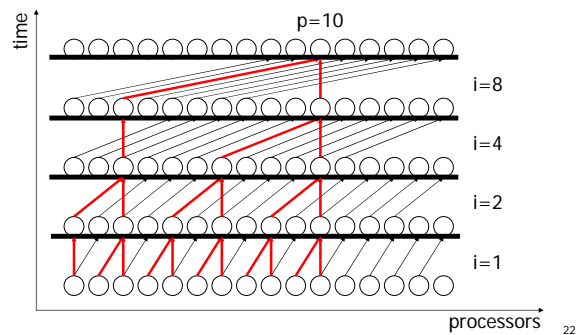
20

Prefix Sums (cont.)



21

Prefix Sums (cont.)



22

Analysis of Prefix Sums

- BSP execution time in general:

$$T = \sum_{s \in \text{super-steps}} (\max_{i \in \text{procs}} w_{i,s} + \max_{i \in \text{procs}} h_{i,s} g + l)$$
- Prefix Sums execution time:
 - Initialization $w_{i,0}=1$
 - All steps perform a "+" operation $w_{i,s}=1$
 - All steps route a 1-relation $h_{i,s}=1$
 - $\lceil \log n \rceil$ super-steps in total
$$T = 1 + \lceil \log n \rceil (1 + g + l)$$

23

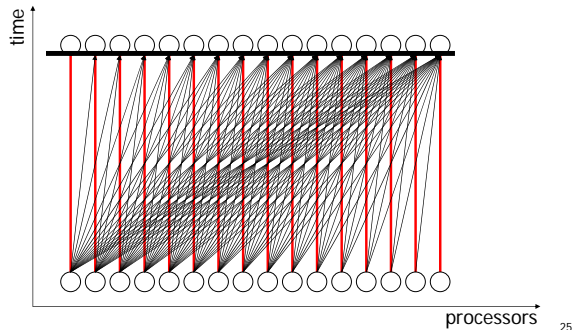
Prefix Sums – Plan B

```

1. for (p=0; p<n; p++) in parallel {
2.   right=init(p); array[0..n-1]=0;
3.   for (i=p+1; i<n; i++)
4.     put(i, right, array[i]);
5.   barrier_synchronize();
6.   for (i=0; i<p; i++)
7.     right=right+array[i];
8. }
```

24

Plan B (cont.)



Analysis of Prefix Sums – Plan B

- Prefix Sums execution time:
 - 2 super-steps, one barrier synchronization
 - Initialization $w_{i,0}=1$
 - Processor $n-1$ performs n "+" operations:

$$\max_{i \in \text{procs}} w_{i,s} = w_{n-1,1} = n$$
 - Processor 0 sends and processor $n-1$ receives $n-1$ messages

$$\max_{i \in \text{procs}} h_{i,s} = h_{0,0} = n-1$$

$$T = 1 + n + (n-1)g + l$$

General Prefix Sums

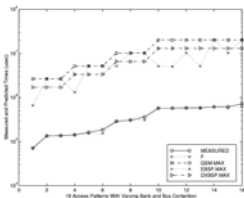
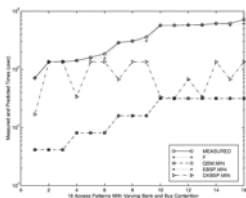
- Assumption $n = P$ (P - number of actual processors) can be dropped using either algorithm – plan A or B:
 1. Sum of array blocks of size n/P computed locally (sequential algorithm)
 2. Use plan A or B to compute the prefix sum in every n/P -th element (last of each block)
 3. Receive the result of the left neighbors prefix sum
 4. Add the received value to the local sums

Design of a BSP program

- Requires machine parameters: l, g, P
 - Analytically derived: too complex, does not work
 - Benchmarks
- Requires computation times of sequential algorithm
 - Analytically derived: too complex, does not work
 - Benchmarks: imprecise since
 - Caching, pipelining effects not repeatable
 - Data dependencies of sequential computation
- In practice: analysis + profiling necessary

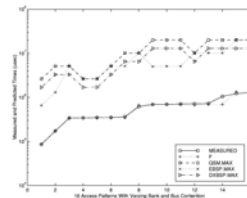
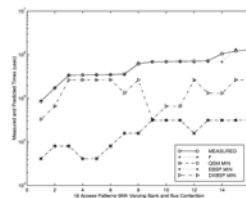
Micro Benchmarks Load

SGI Power Challenge



Micro Benchmarks Store

SGI Power Challenge



Some BSP Machines

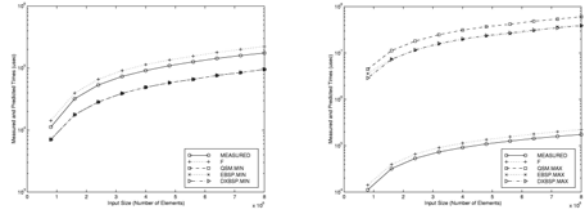
Maschine	<i>l</i>	<i>g</i> (<i>P</i> -relation)	<i>g</i> (1-relation)	<i>P</i>
SGI Power Challenge	25.7	0.13x	0.13x	4
Hitachi SR2001	1321.7	0.92x	0.9x	32
Parsytec GC	6700	34.1x	14.1x	32
DEC-Farm	4664	8.1x		4
IBM SP-2	208.2	0.43x	0.27x	8
Cray T3D	16.6	0.48x	0.36x	32
Cray T3D	31.1	0.78x	0.42x	256

x in words and time in μ s

31

Performance Predictions

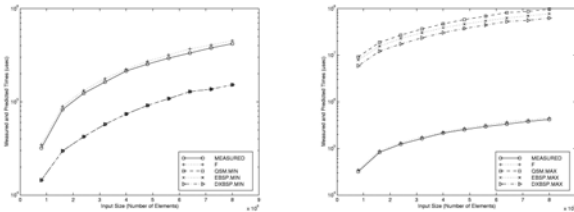
SGI Power Challenge / Radix sort



32

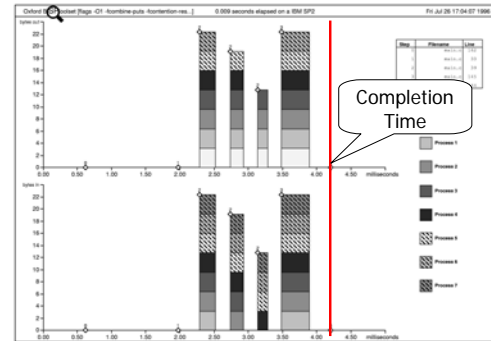
Performance Predictions

SGI Power Challenge / Sample sort



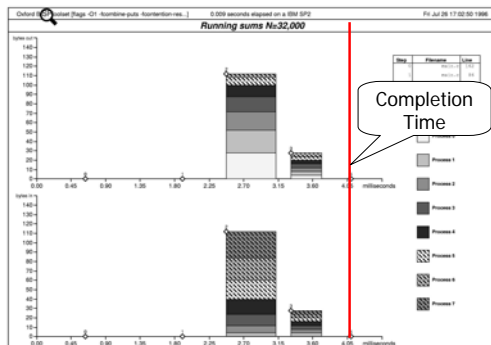
33

Profile Plan A IBM SP/2 8 processors



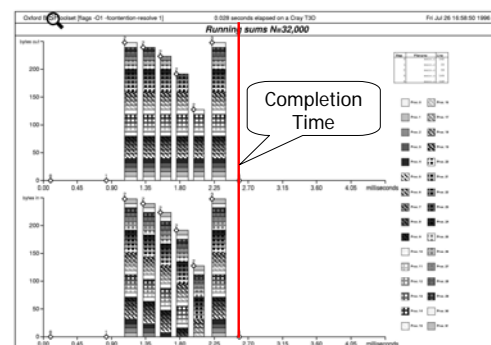
34

Profile Plan B IBM SP/2 8 processors



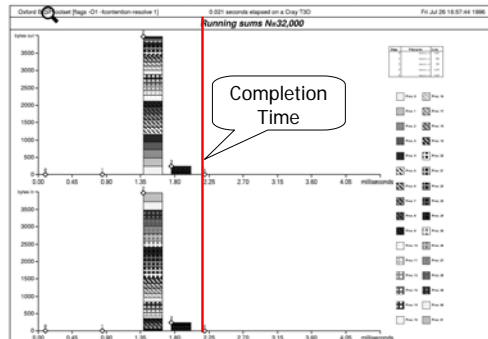
35

Profile Plan A Cray T3D 32 processors



36

Profile Plan B Cray T3D 32 processors



37

Observations

- Plan A could be seen as a PRAM simulation
- Plan B designed directly for BSP
 - Appears absurd on PRAM
 - Advantages show on the more realistic machine model BSP
- Programming becomes more difficult
- Same situation when comparing
 - BSP vs. PRAM
 - PRAM vs. von-Neumann (and parallelization)

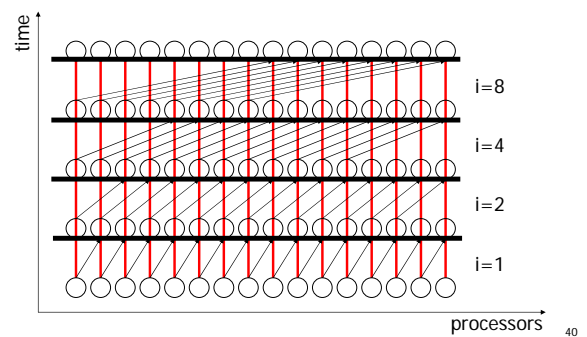
38

Problems with BSP

- Algorithms need to be split in global phases
 - Computation
 - Communication
 - Synchronization
- In many algorithms computation and communication not balanced over processors
- On almost all machines
 - Different times g for local and global communication in a P -relation compared to a I -relation
 - Synchronization
 - Not necessary when knowing all data dependencies,
 - Otherwise, only locally necessary

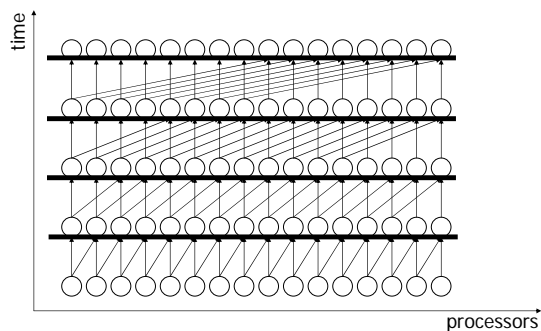
39

Example Prefix Sums (revisited)



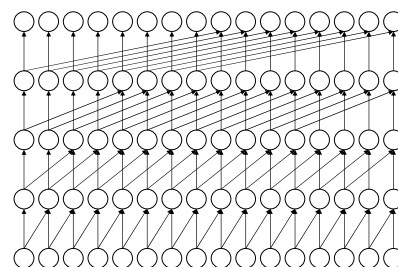
40

Prefix Sums Data Dependencies



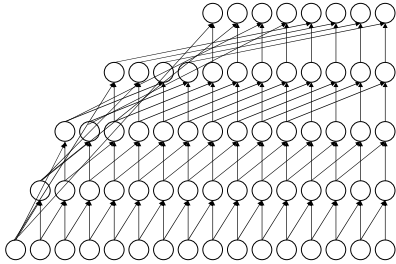
41

Prefix Sums Task Graph



42

Prefix Sums Task Graph



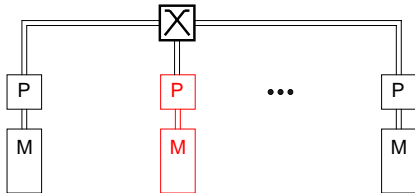
43

Observation

- No barriers required
- Each task
 - Receives required data
 - Performs operations locally
 - Send computed results
- Tasks can be mapped to processors
- Cost model?

44

Delay Model



- Similar architecture like BSP, but
 - infinitely many processors
 - different cost model
- Latency L_i (delay) for a communication of task i
- Computation time w_i of task i

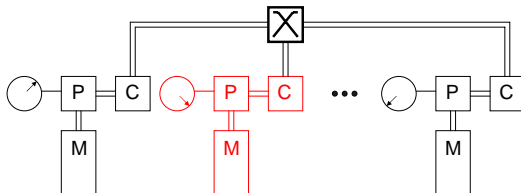
45

Problems

- Too many, too light weight tasks
- Solution:
 - Cluster light weight tasks
 - Schedule clusters to processors
 - We will discuss that later (tomorrow)
- Ignores some actual costs:
 - No bandwidth bound (neither local nor global)
 - No overhead (processor time for communication)

46

LogP (Culler et al.)



- Processor (P), Communication processor (C),
- Memory (M),
- Asynchronous clocks
- Network connected

47

Cost Model for LogP

- For small messages
 - Latency L
 - Overhead of communication o
 - Gap between communications g
 - Capacity bound $\lceil L/g \rceil$
- Number of processors P

48

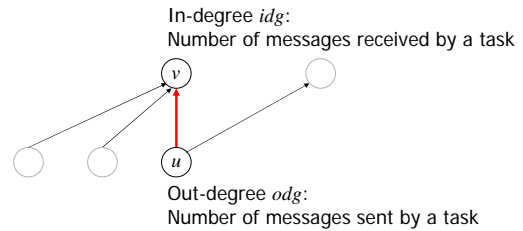
Extended Cost Model for LogP

(Eisenbiegler, Löwe, Zimmermann)

- Functions for modeling the network
- Linear in message size x
 - Latency $L(x) = L_0 + L_1x$
 - Overhead of communication $o(x) = o_0 + o_1x$
 - Gap between communications $g(x) = g_0 + g_1x$
 - Capacity bound $\lceil L_0/g_0 \rceil$
- Number of processors P

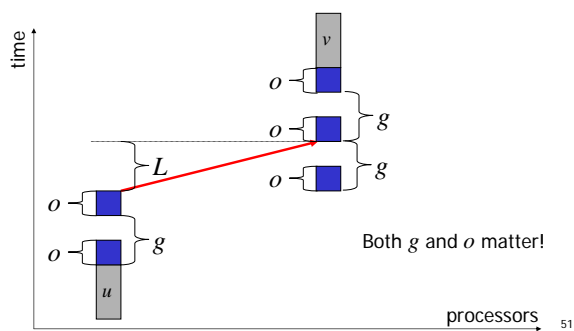
49

Communicating Tasks

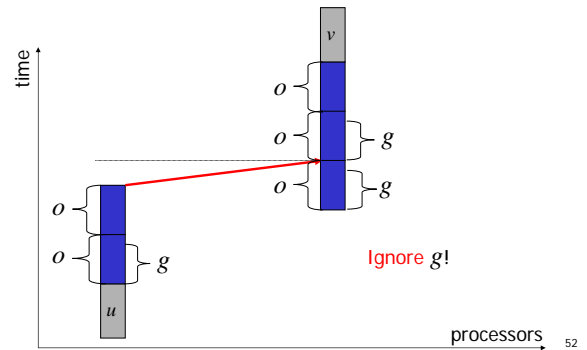


50

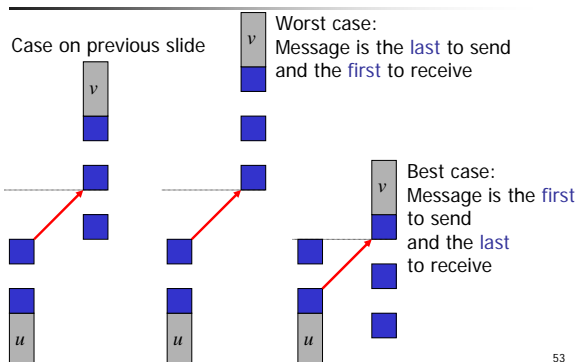
LogP Communication $g > o$



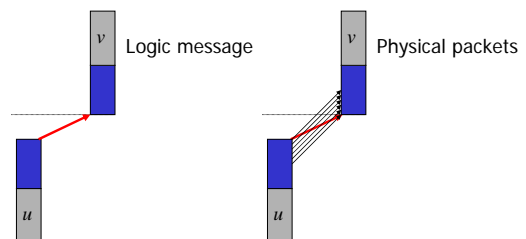
LogP Communication $g < o$



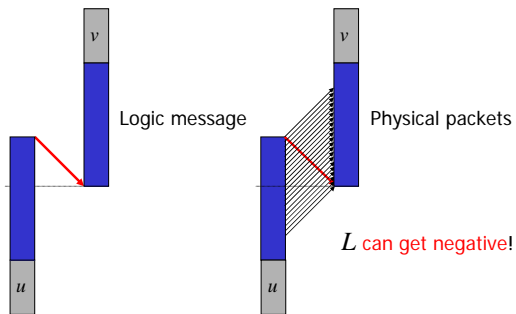
Order of send/receives matters



Messages vs. Packets



Long Messages

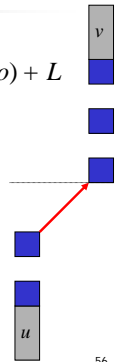


55

Conservative approximation

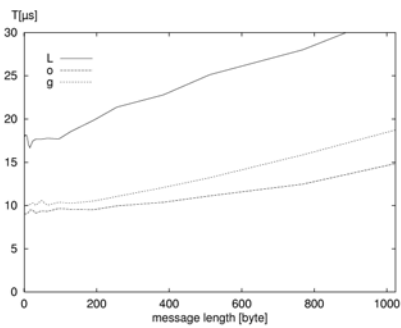
- Time for communication:

$$L_{max}(u,v) = 2o + (odg(u) + idg(v) - 2) \max(g, o) + L$$
- L can become negative
- L, o, g functions in the message size
- Actual communication could be considerably shorter



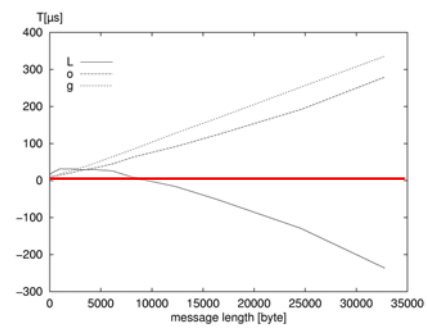
56

Benchmarks IBM SP2 (128 processors)



57

Benchmarks IBM SP2 (128 processors)



58

Some LogP Machines

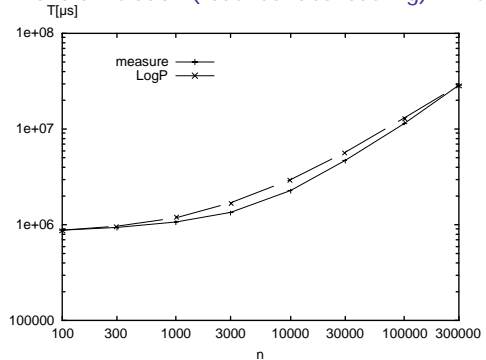
Maschine	L	o	g	P
CM-5	6	2.2	4	512
Meiko CS-2	8.6	1.7	$14.2 + 0.03x$	64
Power Xplorer	$21 - 0.82x$	$70 + x$	$115 + 1.43x$	8
Para-Station	$50 - 0.10x$	$3 + 0.112x$	$3 + 0.119x$	4
IBM SP-2	$13 - 0.005x$	$8 + 0.008x$	$10 + 0.01x$	128
IBM SP-2	$17 - 0.005x$	$8 + 0.008x$	$10 + 0.01x$	256

x in bytes and time in μs

59

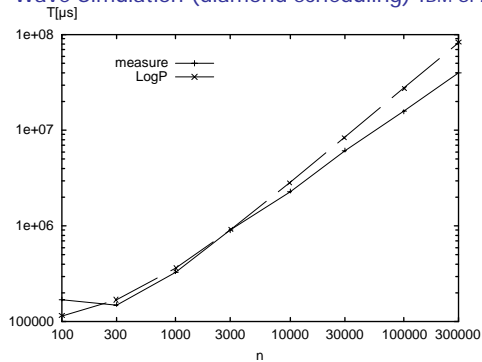
Measurements and Predictions

Wave Simulation (redundant scheduling) IBM SP2



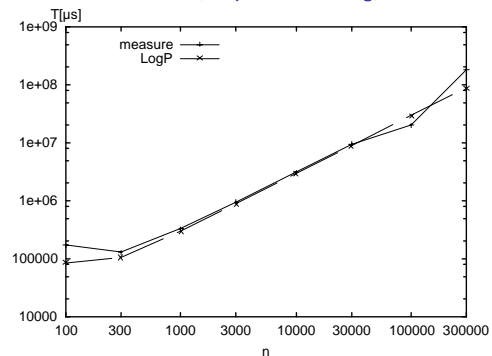
Measurements and Predictions

Wave Simulation (diamond scheduling) IBM SP2



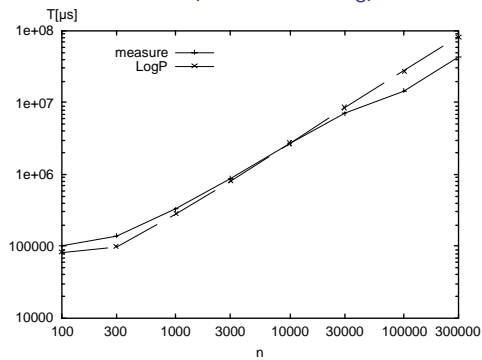
Measurements and Predictions

Wave Simulation (stripe scheduling) IBM SP2



Measurements and Predictions

Wave Simulation (block scheduling) IBM SP2



Example Prefix Sums

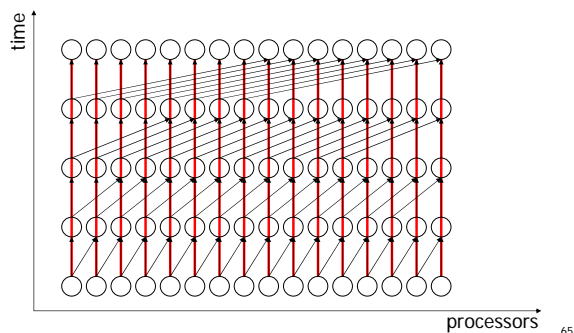
```

1. Process(p) { //pe[0..n-1]
2.   right=p; left=0;
3.   for (i=1; i<n; i*=2) {
4.     if (p+i < n)
5.       send(p+i, right);
6.     if (p >= i) {
7.       left=receive(p-i);
8.       right=right+left;
9.     }
10.  }
11. }

```

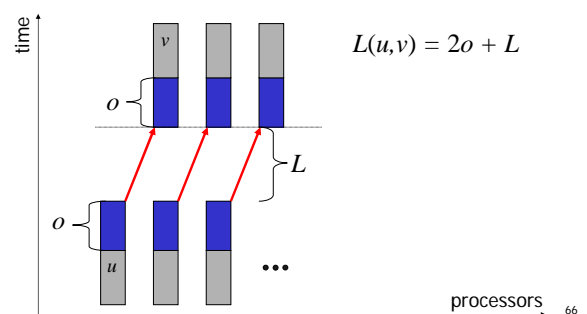
64

Prefix Sums LogP Processes



65

Prefix Sums Communication Step



66

Analysis of Prefix Sums

- Prefix Sums execution time:
 - Initialization $w=1$
 - All loop steps perform a "+" operation $w=1$
 - All loop steps send/receive at most 1 message
 $L(u,v) = 2o + L$
 - Disregard g as long as $g \leq 1 + 2o + L$
 - $\lceil \log n \rceil$ loop steps in total

$$T \leq 1 + \lceil \log n \rceil \max(1 + 2o + L, g)$$

67

Comparison of BSP and LogP

- Prefix sums
- IBM SP-2, small messages (< 16 bytes), $P=16$
 - BSP: $l = 502 \mu s$ $g = 30.1 \mu s$
 - LogP: $L = 17.1 \mu s$ $o = 9.0 \mu s$ $g = 9.8 \mu s$
- $$T_{\text{BSP}} = w + \lceil \log n \rceil (w + 1g + l)$$

$$= \lceil \log n \rceil (w + 1) + \lceil \log n \rceil (g + l)$$

$$= \lceil \log n \rceil (w + 1) + 532.1 \lceil \log n \rceil \mu s$$
- $$T_{\text{LogP}} = w + \lceil \log n \rceil \max(w + 2o + L, g)$$

$$= \lceil \log n \rceil (w + 1) + \lceil \log n \rceil (2o + L)$$

$$= \lceil \log n \rceil (w + 1) + 36.1 \lceil \log n \rceil \mu s$$

68

Design a LogP program

- Execution time is the time of the slowest process
- Implications for algorithm design:
 - Balance computation
 - Balance communications
 are only **subgoals!**
- But mind the capacity constraint $\lceil L_0/g_0 \rceil$
- Avoid communications or at least P2P communications

69

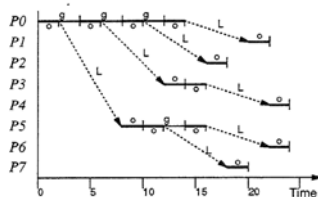
Example Broadcast (Karp et al.)

- Distribute a single item to P processors
- Define a broadcast tree
 - Infinite tree
 - Nodes labeled with times
 - Root gets label 0
 - i -th child of node labeled with t gets label $t + 2o + L + i \times \max(o, g)$ (start counting with child 0)
- Subtree with the P smallest labels induces the optimum broadcast

70

Example Broadcast

$L = 6, o = 2, g = 4, P = 8$



i -th child of node t gets $t + 2o + L + i \times \max(o, g)$

71

LogGP Model (Alexandrow et al.)

- New parameter G for gap per byte in long messages
- Models higher bandwidth for larger packages
- Usually $G \ll g$
- Simplification of LogP model with functions for the parameters
- May be simpler but still adequate

72

Classical LogP vs. LogGP

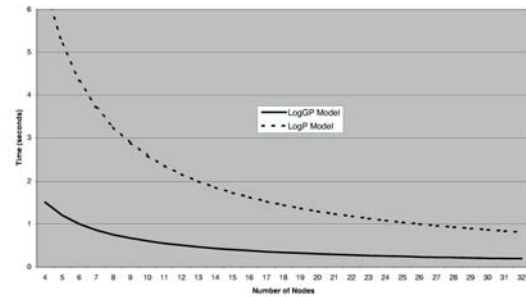
Example: point 2 point k bytes communication

- Communicate k bytes point 2 point
- Classic LogP: $2o + (k-1)\max(o,g) + L$
- LogGP: $2o + (k-1)G + L$
- LogP with functions: $2o(k) + L(k)$

73

Classic LogP vs. LogGP

Network Of Workstations: uniform array distribution



74

Problem with Analyses

- Algorithms and runtime predictions based on global completion times
- LogP architecture and real architectures do not have a general clock
- Prediction–measurement–agreement
 - Obviously it works in practice
 - Why, how ...?

75

LogP with Disturbances

(Löwe, Zimmermann)

- Probability model
 - If a computation/communication could happen in the deterministic model it happens only with a certain probability q
 - q models asynchrony of clocks
- With high probability the delay is only a constant factor
- It holds for any constants c

$$\Pr[T_{\text{async}} > 5c/q((1+\log P) T_{\text{sync}} + \log P)] \leq P^{-c}$$

$$E[T_{\text{async}}] \leq 6/q((1+\log P) T_{\text{sync}} + \log P)$$

76

Problems with LogP

- Programming gets more complicated with LogP than with BSP or PRAM
- Depending on the cases
 - it is worth the effort
 - it depends on the problem size
 - it does not pay at all
- Tell one situation from the others
- Find simulations
- Find automatic transformations

77