CS 498
Hot Topics in High Performance Computing

Networks and Fault Tolerance

5. Advanced Network Models

Intro

- What did we learn in the last lecture
  - The LogP model and examples (more broadcasts)
  - Analyzing a parallel Fast Fourier Transform in LogP
- What will we learn today
  - Continue Fast Fourier Transform in LogP
  - LogGP a first LogP extension
    - The Scatter Problem
  - LogGPS a second LogP extension

Algorithm Design: FFT

- Assuming n (power of 2) inputs and butterfly radix-2 FFT DAG (Cooley&Tukey)
- DAG has n(log n+1) nodes arranged in n rows and log n+1 columns
- For $0 \leq r < n$ and $0 \leq c < \log(n)$,vertex (r,c) has edges to vertex (r,c+1) and (r_c',c+1) where r_c' is determined by negating the (c+1)-th bit in r
- Each non-input node represents a complex operation, each edge communication

Parallel Data Layout

- Block decomposition (w.l.o.g, assuming P%n=0):
  - Assign i-th n/P rows to process i-1
  - First log(P) columns require remote data
  - Last log(n/P) columns require no communication
- Times:
  - T_{comp} = n/P log(n) compute steps
  - T_{comm} = (g*n/P+L) log(P) (assuming g>2o [1])

[1]: Culler et al.: “LogP: towards a realistic model of parallel computation”

Parallel Data Layout

- Cyclic distribution (w.l.o.g, assuming P%n=0):
  - Assign i-th row to process i%P
  - First log(n/P) columns require no communication
  - Last log(P) columns require remote data
- Times:
  - T_{comp} = n/P log(n) compute steps
  - T_{comm} = (g*n/P+L) log(P) (assuming g>2o [1])

[1]: Culler et al.: "LogP: towards a realistic model of parallel computation"

Optimal Layout?

- Class Question: How would you arrange the n elements on P processes?

Optimal Layout?

- Class Question: How would you arrange the n elements on P processes?
  - Yes, cyclic in the first log(P) columns and block in the last log(P)
    - Switching between log(P)-th and log(n/P)-th stage is fine
  - Single all-to-all communication step (if n/P>P)
  - Each processor sends n/P^2 items to each destination
  - $T_{comm}=L+g(n/P-n/P^2)$
    - more than a factor of log(P) faster than any decomp.!
    - Within a factor of (1+g/log(n)) of optimal!

[1]: Culler et al.: "LogP: towards a realistic model of parallel computation"

Communication Schedule

- We showed a good data arrangement for FFT
  - Need communication schedule that avoids hot spots
  - How to perform the all-to-all communication?
- Variant 1 (naïve):
  - for(int i=0; i<P; ++i) { irecv from i; send to i; }
  - $T_{comm} = P(P-1)g + L$
- Variant 2 (optimized):
  - Class Question!

Communication Schedule

- We showed a good data arrangement for FFT
  - Need communication schedule that avoids hot spots
  - How to perform the all-to-all communication?
- Variant 1 (naïve):
  - for(int i=0; i<P; ++i) { irecv from i; send to i; }
  - $T_{comm} = P(P-1)g + L$
- Variant 2 (optimized):
  - for(int i=0; i<P; ++i) { irecv from (id-i)%P; send to (id+i)%P; }
  - $T_{comm} = (P-1)g + L$

Overlapping Communication and Computation

- if o<<g, CPU idles for g-o cycles between successive transmissions (e.g., all-to-all)
- One can now compute the communication-optimal FFT that overlaps communication and computation!
  - Needs model for FFT computation time
  - Remainder is straight-forward (applying the steps we did before)

Back to Optimal Broadcast

- What did we miss in the previous analysis?
  - Class Question!

Back to Optimal Broadcast

- What did we miss in the previous analysis?
  - Yes, s – we only dealt with a single-packet bcast ☹
- Karp et al. show that k-item broadcasts can be performed in time B(P)+2L+k-2 if B(P) is the time for a single-item broadcast
  - Details in Karp et al.: "Optimal Broadcast and Summation in the LogP Model"
  - We will see that LogP is suboptimal for large messages, thus not look at this in detail!

LogP Benefits over Simpler Models

- Models pipelining effects
  - Modern networks support outstanding messages
  - Leads to better algorithms
- Models CPU overhead
  - Enables analysis of computation/communication overlap
  - Important for complex collective operations
    - E.g., nonblocking collective operations

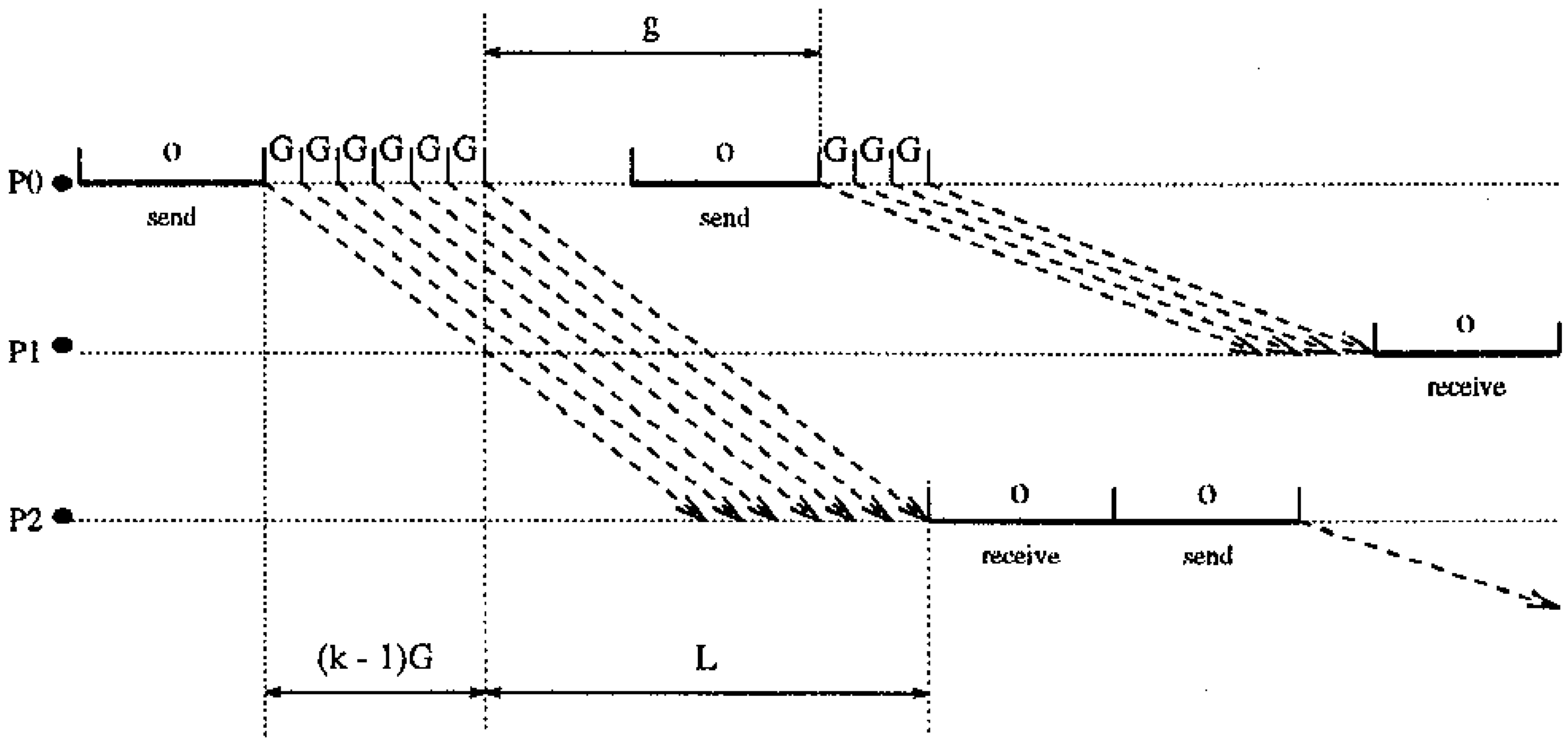
Concerns with LogP

- Is it tractable to analyze non-trivial algorithms?
  - It's much more complex than PRAM or BSP
- The model ignores topology completely
  - Some topologies are well understood, no convergence in architecture though!
- Bulk messaging?
  - HW-supported fragmenting enables fast bulk transmission
- Is MPI modeled detailed enough?

LogGP – A first Extension to LogP

- Extends the basic LogP model with a linear model for large messages
  - G = cost per Byte, reciprocal is the bandwidth
- Models bulk message transfer (packaging and pipelining in hardware)
  - Not every packet is created by the processor
  - Every modern HPC network supports this!
- Changes algorithm design and cost tradeoffs
  - Significantly different algorithms

LogGP Visualization



LogGP Examples

- Sending a single message of size s
  - Class Question
- Ping-Pong Round-Trip of size s
  - Class Question
- Transmitting n messages of size s
  - Class Question

LogGP Examples

- Sending a single message of size s
  - $T(s) = 2o + L + (s-1)G$
- Ping-Pong Round-Trip of size s
  - $T_{RTT}(s) = 4o + 2L + 2(s-1)G$
- Transmitting n messages of size s
  - $T(n,s) = L + (n-1)\max(g, o) + n(s-1)G + 2o$

Some Simple Observations

- Bulk messaging is important for algorithm design!

1. Send largest possible messages!
   – Splitting messages almost never helps (only in complex scenarios such as forwarding)
2. New trade-offs for complex network operations: L/o/g vs. G
   – Will be discussed next

LogGP Motivation: Scatter

- Send different items from one process to each other process (aka personalized broadcast)
- Class Question: What is the optimal algorithm in LogP and what is it’s runtime (assume o=0)?

LogGP Motivation: Scatter

- Send different items from one process to each other process (aka personalized broadcast)
- Class Question: What is the optimal algorithm in LogP and what is it's runtime (assume o=0)?
  - The source sends all (P-1)*s items to their destinations
  - No faster way exists since they all need to leave the source!
  - $T(s) = (s(P-1)-1)g + L$