CS 598: Communication Cost Analysis of Algorithms

Lecture 1: Course motivation and overview; collective communication

Edgar Solomonik

University of Illinois at Urbana-Champaign

August 22, 2016

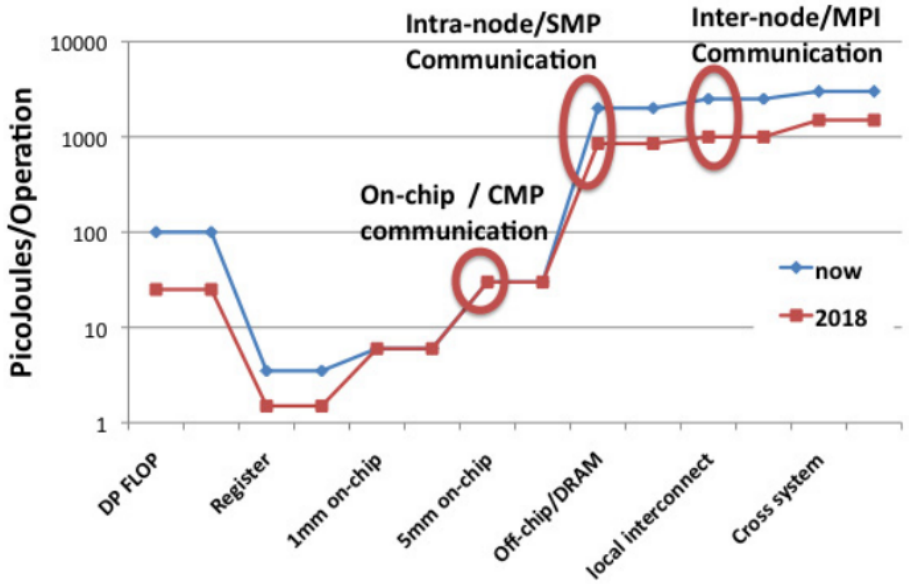
Growth of gaps between flop rate, bandwidth, and latency

Table : Annual improvements

flop rate		bandwidth	latency
59%	DRAM	26%	15%
	Network	23%	5%

- flop rate – number of operations on register-resident elements a CPU can perform per second
- bandwidth – amount of data moved per second
- latency – time between request and reception of one datum

Energy gaps between flop rate and bandwidth



10 Teraflop chip design requirements based on above

- 100 Watts for floating point units
- 2000 Watts for memory bandwidth for floating point ratio of 0.2

"The consequence is that we can engineer far more floating point capability onto a chip than can reasonably be used by an application. **Engineering FLOPs is not a design constraint – data movement presents the most daunting engineering and computer architecture challenge.** " – Shalf, Dosanjh, Morrison, VECPAR 2010

Types of distributed systems

Modern computer systems handling intensive workloads are most often distributed memory

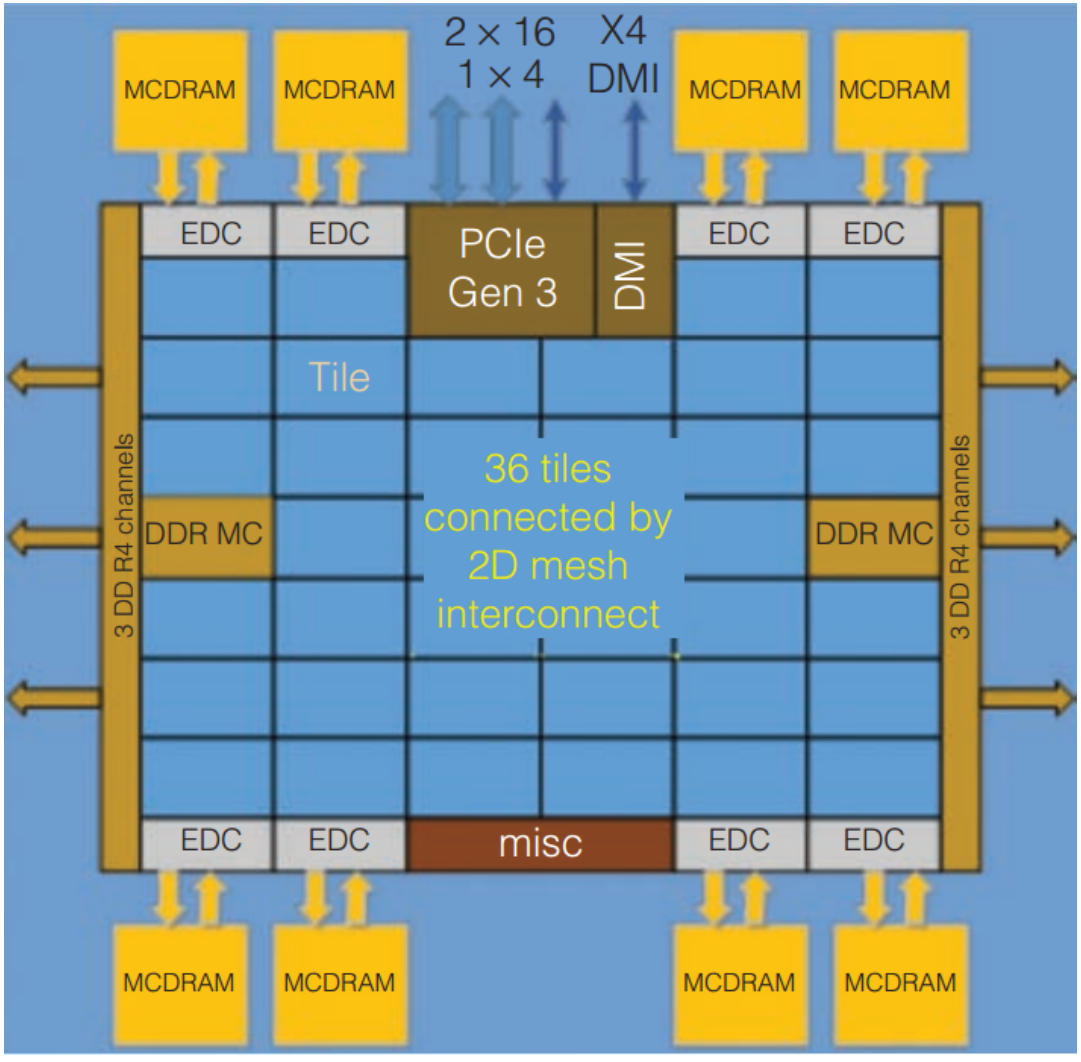
- cloud computing (servers)
- mobile computing
- supercomputing

However, the problems these systems target may differ greatly

- this course will focus on computations arising in scientific applications and data analysis
- designing communication-efficient algorithms is also important for other workloads (e.g. streaming requests)

Distributed memory system on a chip

Intel Knight's Landing chip layout



Algorithms, schedules, and optimizations

The course will focus on algorithms rather than scheduling optimizations

- overlap between communication and computation will not improve performance by more than 2X
  - caveat: overlap between many cost components (e.g. internode and intranode communication and synchronization) can yield speed-ups up to the number of components
- topology-aware mapping is system-specific and not always possible
- algorithmic improvements provide potential for asymptotic improvements in performance and scalability

Algorithms versus schedules

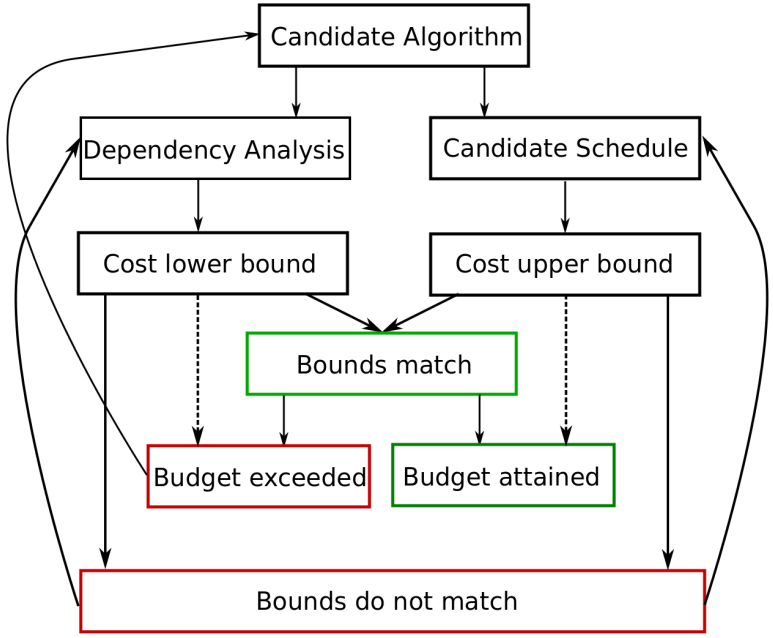
- "parallel algorithm" and "schedule" are often used interchangeably, we will try to be more precise
- algorithm – family of dependency graphs
- schedule – parallelization and communication specification

Algorithms: addressing the problem at its root

Parallelizability is best achieved a priori

- "design to be parallelizable" rather than "design then parallelize"
- the space of algorithms for a problem is usually richer than the space of schedules for an algorithm

Communication lower bounds provide a rigorous view of the space of schedules for an algorithm



Success story: all-pairs shortest paths (APSP)

Classical algorithm: Floyd-Warshall

- $O(n^3/p)$ computation per processor
- $O(n^2/p^{2/3})$ data sent/received per processor
- $O(p^{2/3})$ messages sent/received per processor

Lower bound analysis shows that Floyd-Warshall cannot be parallelized more efficiently

A more parallelizable approach: path doubling

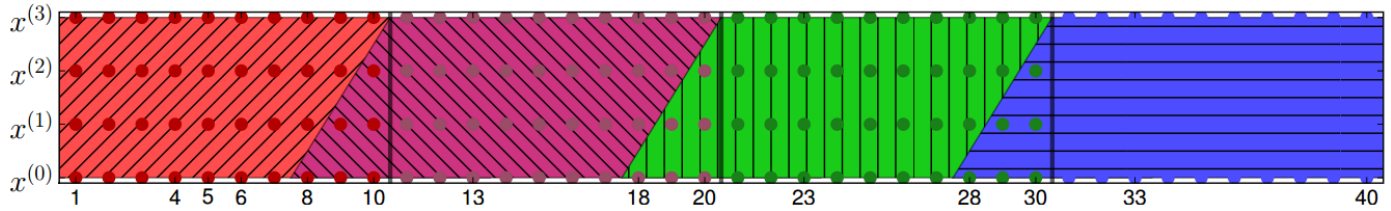
- $O(n^3/p)$ computation per processor
- $O(n^2/p^{2/3})$ data sent/received per processor
- $O(\log p)$ messages sent/received per processor

Tiskin, Alexander. "All-pairs shortest paths computation in the BSP model.", 2001.

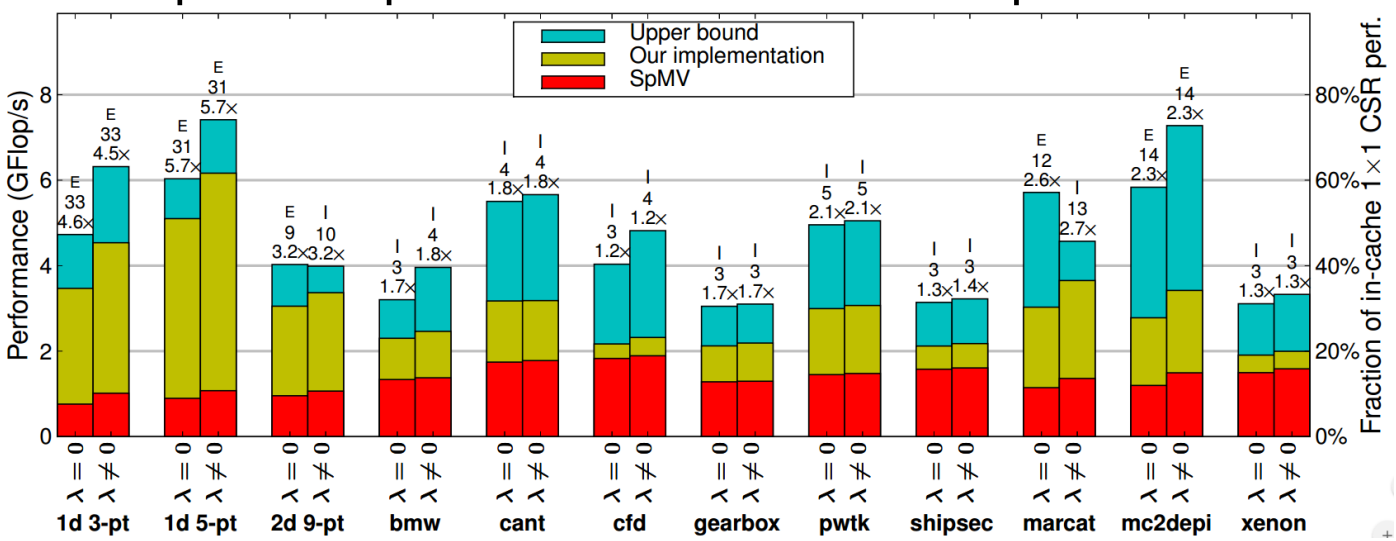
We will revisit and study this algorithm in detail in the course.

Success story: sparse iterative methods

Scheduling idea for simple stencil computations: $x_j^{(i+1)} = f(x_{j-1}^{(i)}, x_j^{(i)})$



Generalizes to repeated sparse matrix-vector multiplications $x^{(i+1)} = Ax^{(i)}$



The idea has motivated design of new numerical algorithms and has also been studied via lower bound analysis.

Mohiyuddin, Marghoob, et al. "Minimizing communication in sparse matrix solvers.", 2009.

We will study the algorithm and relevant lower bounds later in the course.

Outline of course [1]

- communication cost models
  - different messaging models (BSP, LogP, LogGP) and models for fast/slow memory
  - collective communication protocols highlight differences and similarities
  - for most of the course we will work with a basic $\alpha - \beta$ messaging model
- algorithms for FFT, permutation networks, sorting will be surveyed
- communication avoidance in dense linear algebra
  - communication-optimal distributed-memory algorithms
  - will go into more depth on recent results than CS 554 (`https://courses.engr.illinois.edu/cs554/fa2015/notes/index.html` still a great supplementary reference for this course)
- sparse iterative methods
  - *s*-step methods (in-time blocking) will receive most attention
  - other techniques, including asynchronous methods, will be surveyed
- graph algorithms
  - will cover shortest-path algorithms in depth
  - many analogous ideas to parallel algorithms for linear algebra
  - other problems and algorithms will be surveyed

[1] The ordering of and depth with which topics are covered might change.

Outline of course

Topics covered in the second half of the semester will depend on project presentation schedule

- communication lower bounds
  - classical Jia-Wei & Kung pebbling lower bounds
  - more recent general techniques: representations of algorithms as graphs and tensor factorizations, expansion analysis
- multidimensional data analysis: tensor contraction and factorization
- avoiding communication in (scientific) applications
  - optimal partitioning strategies for particle-codes (molecular dynamics)
  - electronic structure methods (time-permitting)
  - potential special topics: multigrid, convolutional neural networks
- network topologies and topology-aware algorithms will be surveyed (time-permitting)
  - potential lecture on Slim-Fly topology

Lectures

Borrowing lecture structure from John Kubiatowicz (UC Berkeley)

- 30-35 minutes technical
- 3-10 minutes break/discussion
- 5 minutes administrative
- 30-35 minutes technical

Homeworks

- weekly short assignments, Wednesday to start of class Wednesday
- electronic submission via email (solomon2@illinois.edu)
- answers posted Friday at noon, graded by Friday evening
- 11 in total (depending on project presentation schedule)
- 10 points each (equal weight)
- 1 lowest grade dropped
- 1 can be turned in late (by Friday at noon) with no penalty
- 25% penalty if turned in by Thursday at noon
- 50% penalty if turned in by Friday at noon
- penalties rounded down to half point
- should be completed independently [2]

[2] Cheating policy: academic integrity will be in line with university policy, *minimum* penalty on first attempt - 0% on assignment and lowest grade not dropped

Course resources

- Piazza: CS 598 ES
  - good for discussion of lecture topics, relevant references, questions
  - questions and answers related to homework problems are ok, but providing homework answers can violate academic integrity
- email me questions directly if they are not of general interest or require discussion of answers to homework problems
- webpage: `http://solomon2.web.engr.illinois.edu/teaching/cs598_fall2016/index.html`
- office hours (room TBA):
  - (option 1) Monday 11-12
  - (option 2) Friday 12-1
- references posted on course website and throughout lectures

Project

- should endeavor to obtain a novel result
- theoretical component required, but does not have to be predominant
- individual presentation and report
- joint work and shared results possible, but individual contributions must be clear
- proposal stage 1 due to Sep 21, stage 2 (revision) due Oct 19 (no homeworks those weeks)

Typical acceptable projects

- propose and analyze a complex new algorithm or schedule
- propose and implement a simple new algorithm or schedule
- implement and analyze a known algorithm or schedule
- do a deep literature survey and analyze a complex known algorithm (i.e. if you take-on a risky project, there will be flexibility and potential for fallback)

Overall grading

- Option 1:
  - 50% homework
  - 40% project
  - 5% 1st proposal
  - 5% 2nd proposal
- Option 2:
  - 20% test (with focus on general understanding) on Nov 16
  - 35% homework (two fewer homework assignments)
  - 35% project
  - 5% 1st proposal
  - 5% 2nd proposal

Components (homeworks/tests) or letter grades may be curved upward

Transition

Course material begins here!

First, lets discuss how to quantify communication and synchronization.

A simple model for point-to-point messages

The time to send or receive a message of s bytes is

$$T_{\text{sr}}^{\alpha,\beta}(s) = \alpha + s \cdot \beta$$

- α – **latency/synchronization cost** per message
- β – **bandwidth cost** per byte
- each processor can send and/or receive one message at a time

Let P processors send a message of size s in a ring,

- the **communication volume** (total amount of data sent) is $P \cdot s$
- What is the **communication cost** (α–β-model execution time)?
  - if the messages are sent *simultaneously*,

$$T_{\text{sim}-\text{ring}}^{\alpha,\beta}(s) = T_{\text{sr}}^{\alpha,\beta}(s) = \alpha + s \cdot \beta$$

  - if the messages are sent *in sequence*,

$$T_{\text{seq}-\text{ring}}^{\alpha,\beta}(s, P) = P \cdot T_{\text{sr}}^{\alpha,\beta}(s) = P \cdot (\alpha + s \cdot \beta)$$

Broadcasts in the α–β model

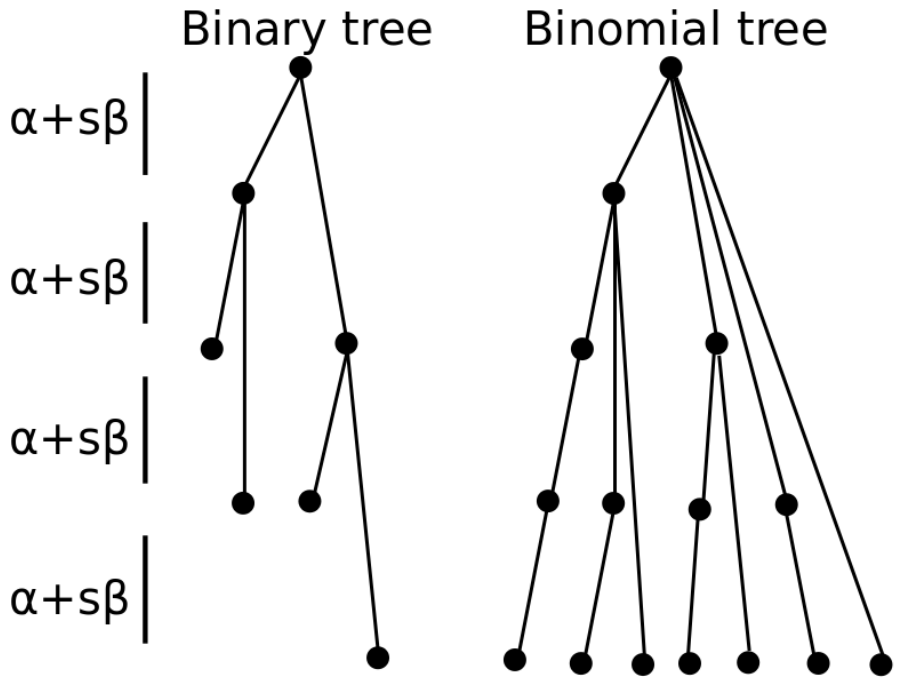
The execution time of a broadcast of a message of size s to P processors is

- using a **binary tree** of height
  $h_r = 2(\log_2(P+1)-1) \approx 2\log_2(P)$

  $$T^{\alpha,\beta}_{\text{bcast}-\text{bnr}}(s,P) = h_r \cdot T^{\alpha,\beta}_{\text{sr}}(s)$$
  $$= h_r \cdot (\alpha + s \cdot \beta)$$

- using a **binomial tree** of height
  $h_m = \log_2(P+1) \approx \log_2(P)$

  $$T^{\alpha,\beta}_{\text{bcast}-\text{bnm}}(s,P) = h_m \cdot T^{\alpha,\beta}_{\text{sr}}(s)$$
  $$= h_m \cdot (\alpha + s \cdot \beta)$$



Therefore, a binomial tree broadcast is $h_r/h_m \approx 2$ faster than a binary tree broadcast in the α–β model

Large-message broadcasts

Lets now consider broadcasts of a message of a size $s \geq P$ bytes

- recall binomial tree broadcast cost:

$$T^{\alpha-\beta}_{\mathrm{bcast-bnm}}(s,P) = \log_2(P+1) \cdot (\alpha + s \cdot \beta)$$

- consider instead the following broadcast schedule
  - the root sends a different segment of the message to each processor
  - all processors exchange segments in $P-1$ near-neighbor ring exchanges
- the cost of this broadcast schedule is

$$\begin{aligned} T^{\alpha-\beta}_{\mathrm{bcast-ring}}(s,P) = & \ (P-1)(T^{\alpha-\beta}_{\mathrm{sr}}(s/P) + T^{\alpha-\beta}_{\mathrm{sim-ring}}(s/P)) \\ & = 2(P-1)(\alpha + s/P \cdot \beta) \approx 2(P \cdot \alpha + s \cdot \beta) \end{aligned}$$

- for sufficiently large message sizes, the new schedule is faster,

$$\lim_{s\to\infty} \left(\frac{T^{\alpha-\beta}_{\mathrm{bcast-bnm}}(s,P)}{T^{\alpha-\beta}_{\mathrm{bcast-ring}}(s,P)} \right) \approx \log_2(P)/2$$

Optimal broadcasts?

We will study a series of protocols for collectives, including broadcasts

- lets first do a bit of foreshadowing...
- from here on, let $h = \log_2(P) \approx \log_2(P+1)$
- the fastest broadcast protocol we will consider, (due to Träff and Ripke, 2008) has a running time of

$$T^{\alpha-\beta}_{\text{bcast}-\text{TR}} = (\sqrt{h \cdot \alpha} + \sqrt{s \cdot \beta})^2 \leq 2(h \cdot \alpha + s \cdot \beta)$$

- is this optimal? ... open question!
- here is a trivial lower bound (think about it on your own)

$$T^{\alpha-\beta}_{\text{bcast}} \geq h \cdot \alpha + s \cdot \beta$$

  this can be a factor of two less than the above upper bound
- if one restricts all messages to be of a given size, a tight lower bound can be obtained (see Sanders and Sibeyn, 2003)

$$T^{\alpha-\beta}_{\text{bcast}-\text{restricted}} \geq T^{\alpha-\beta}_{\text{bcast}-\text{TR}}$$

Pipelined binary tree broadcast

Send a packet of size k to left child then to right child

- as before, total message size s, tree height $h \approx \log_2(P)$
- each message costs $\alpha + k \cdot \beta$
- root sends $2s/k$ messages
- last packet takes $2h$ sends to reach rightmost tree leaf
- therefore, the total cost expression is

$$\begin{aligned} T^{\alpha,\beta}_{\mathrm{PBT}}(s,P,k) &\approx 2(h + s/k)(\alpha + k \cdot \beta) \\ &= 2(h \cdot \alpha + s \cdot \beta + (s/k) \cdot \alpha + hk \cdot \beta) \end{aligned}$$

- we can now derive the optimal message size

$$k^{\alpha,\beta}_{\mathrm{opt}}(s,P) = \operatorname*{argmin}_k(T^{\alpha,\beta}_{\mathrm{PBT}}(s,P,k)) = \sqrt{\frac{s \cdot \alpha}{h \cdot \beta}}$$

- furthermore, $T^{\alpha,\beta}_{\mathrm{PBT}}(s,P,k^{\alpha,\beta}_{\mathrm{opt}}(s,P)) \approx 2(\sqrt{h \cdot \alpha} + \sqrt{s \cdot \beta})^2$, a factor of 2 more expensive than the Träff and Ripke protocol

Double Tree

Observation: *the leaves of a binary tree,* $(P-1)/2$ *processors, send nothing, while the internal nodes do all the work.*

Double Pipelined Binary Tree Broadcast

- define two pipelined binary trees with a shared root
- non-root processors act as a leaf in one and as an internal node in the second
- send half of the message down each tree, alternating directions with packets of size k

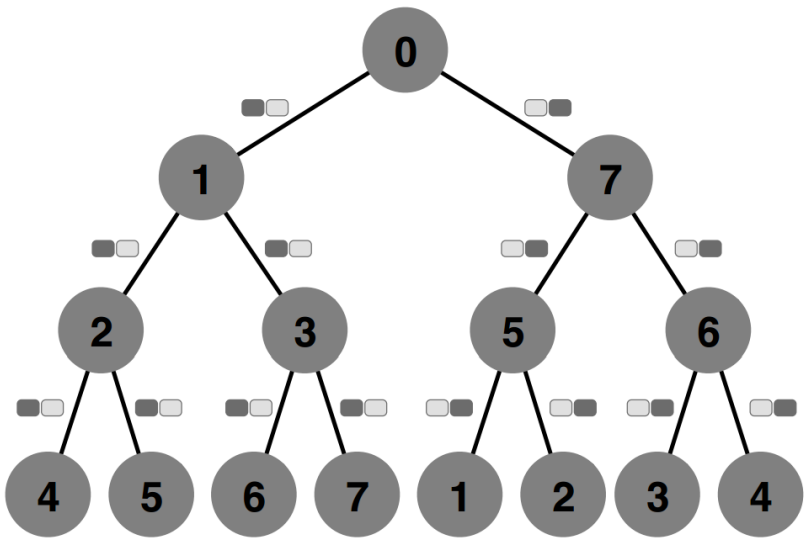


Diagram taken from: Hoefler, Torsten, and Dmitry Moor. "Energy, Memory, and Runtime Tradeoffs for Implementing Collective Communication Operations."

Double pipelined binary tree

The cost of the double pipelined binary tree is essentially the same as the cost of a single pipelined binary tree with half the message size,

$$T^{\alpha,\beta}_{\mathrm{DPBT}}(s,P) \approx 2h \cdot \alpha + 2\sqrt{2s \cdot h} \cdot \sqrt{\alpha \cdot \beta} + s \cdot \beta$$

for a sufficiently large message size (s) this is twice as fast as a single pipelined binary tree.

Other types of collective communication

We can classify collectives into four categories

- **One-to-All**: Broadcast, Scatter
- **All-to-One**: Reduce, Gather
- **All-to-One + One-to-All**: Allreduce (Reduce+Broadcast), Allgather (Gather+Broadcast), Reduce-Scatter (Reduce+Scatter), Scan
- **All-to-All**: All-to-all

MPI (Message-Passing Interface) provides all of these as well as variable size versions (e.g. (All)Gatherv, All-to-allv), see online for specification of each routine.

We now present protocols for these and their cost in the $\alpha - \beta$ model, with

$$s = \begin{cases} \text{input size} & : \text{one-to-all collectives} \\ \text{output size} & : \text{all-to-one collectives} \\ \text{per-processor input/output size} & : \text{all-to-all collectives} \end{cases}$$

Tree collectives

We have demonstrated how (double/pipelined) binary trees and binomial trees can be used for broadcasts

- *A reduction may be done via any broadcast tree with the same communication cost, with reverse data flow*

$$T_{\text{reduce}} = T_{\text{broadcast}} + \text{cost of local reduction work}$$

Scatter is strictly easier than broadcast, pipeline half message to each child in a binary tree

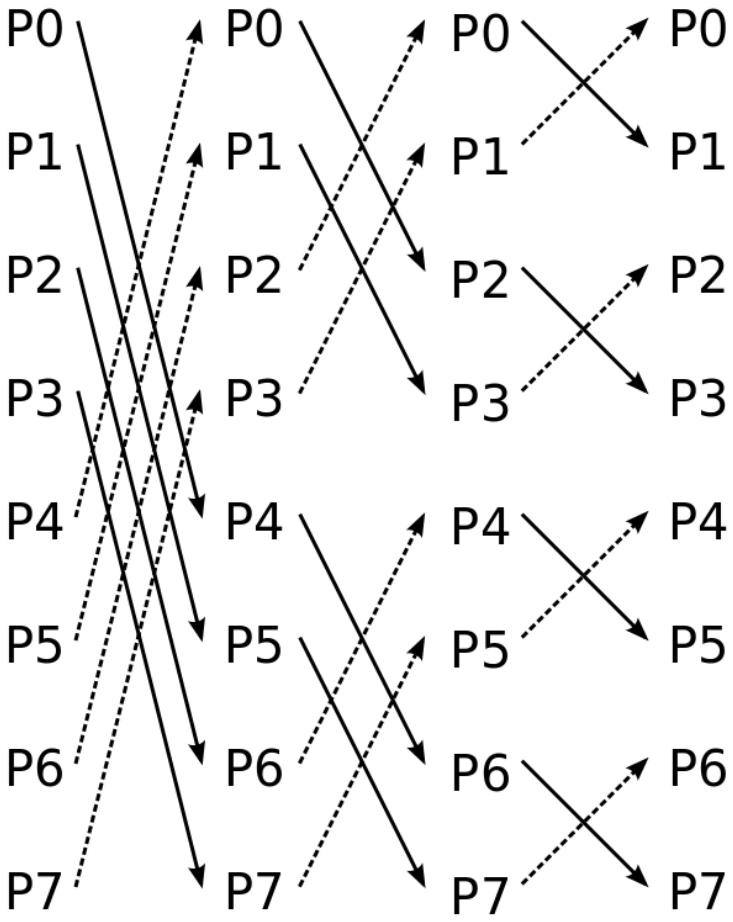
$$T^{\alpha,\beta}_{\text{scatter}}(s, P) \approx 2\log_2(P) \cdot \alpha + s \cdot \beta$$

- *A gather may be done via the reverse of any scatter protocol*:

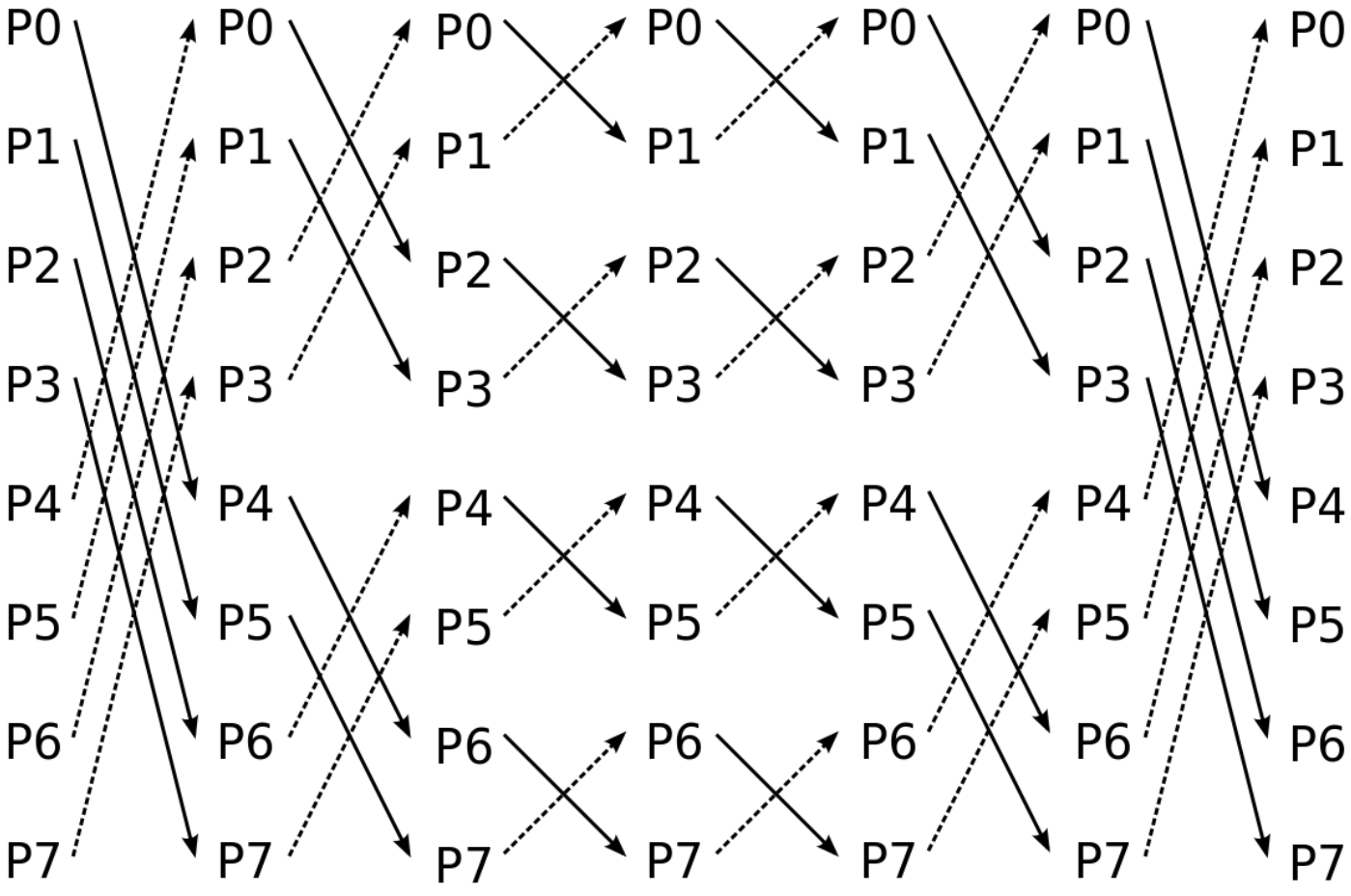
$$T_{\text{gather}} = T_{\text{scatter}}$$

All-to-One + One-to-All collectives can be done via two trees, but is this most efficient? What about **All-to-All** collectives?

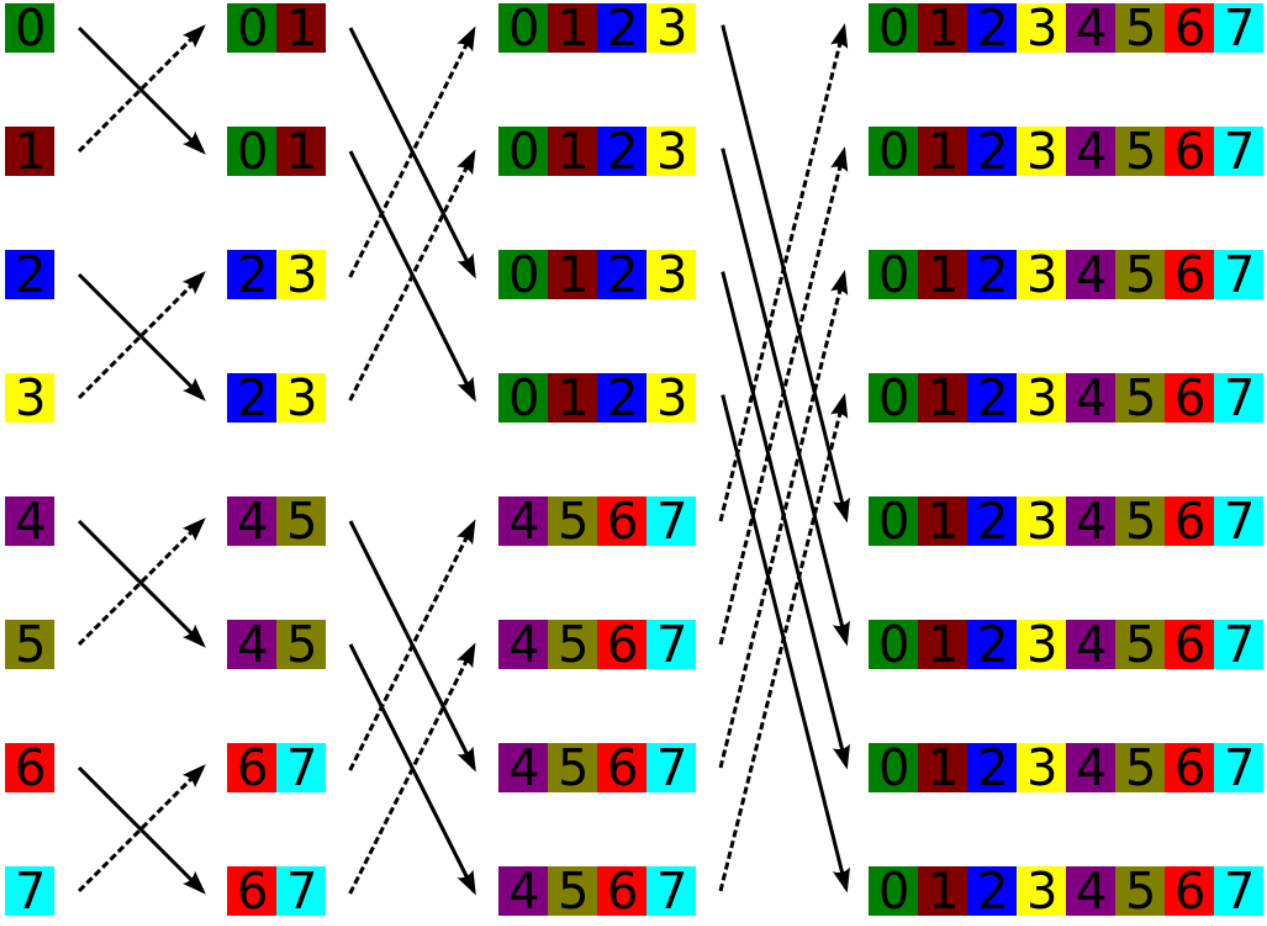
Butterfly network



Butterfly network



Butterfly Allgather (recursive doubling)



Cost of butterfly Allgather

The butterfly has $\log_2(P)$ levels. The size of the message doubles at each level until all s elements are gathered, so the total cost is

$$T^{\alpha,\beta}_{\text{allgather}}(s,P) = \begin{cases} 0 & : P = 1 \\ T^{\alpha,\beta}_{\text{allgather}}(s/2, P/2) + \alpha + (s/2)\cdot\beta & : P > 1 \end{cases}$$

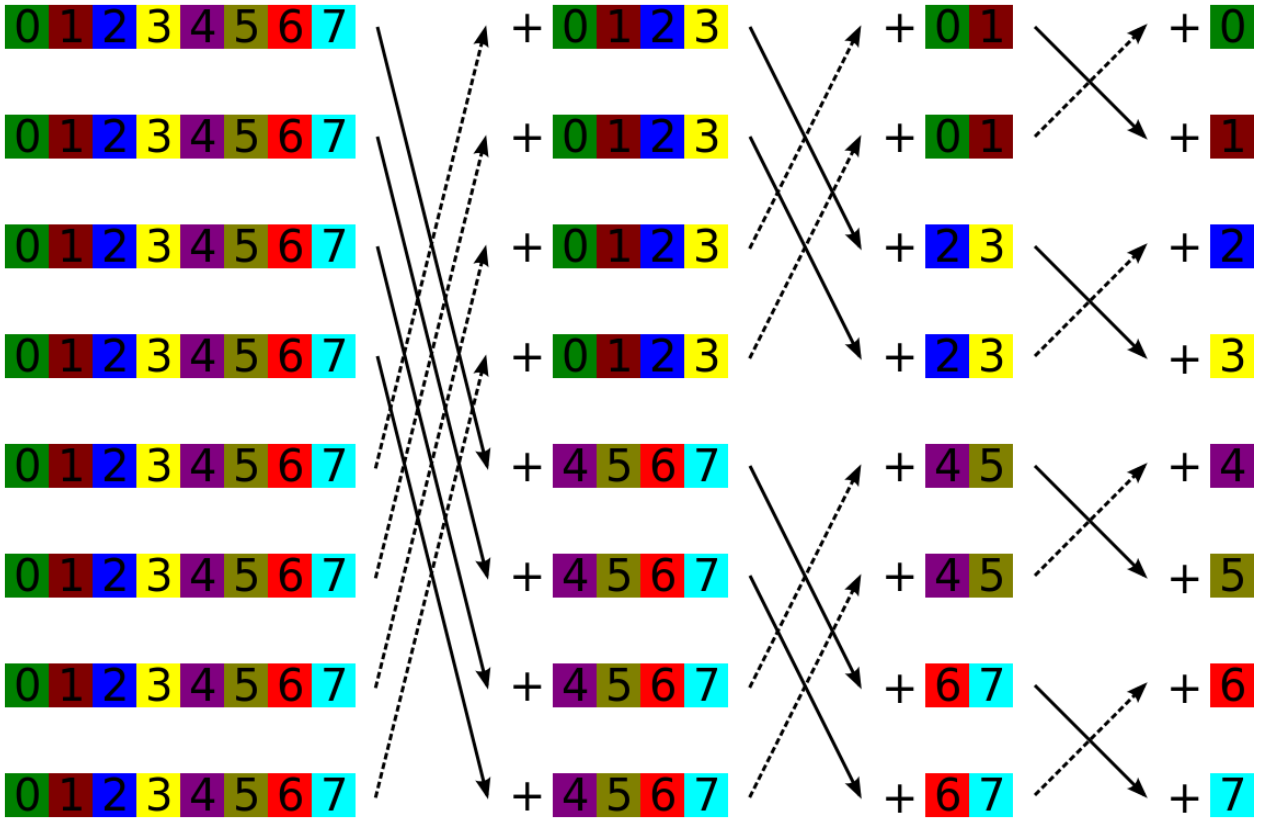
$$\approx \log_2(P)\cdot\alpha + \sum_{i=1}^{\log_2(P)} s/2^i \cdot \beta$$

$$\approx \log_2(P)\cdot\alpha + s\cdot\beta$$

The geometric summation in the cost is characteristic of one-to-all, all-to-one, and all-to-one-to-all butterfly protocols

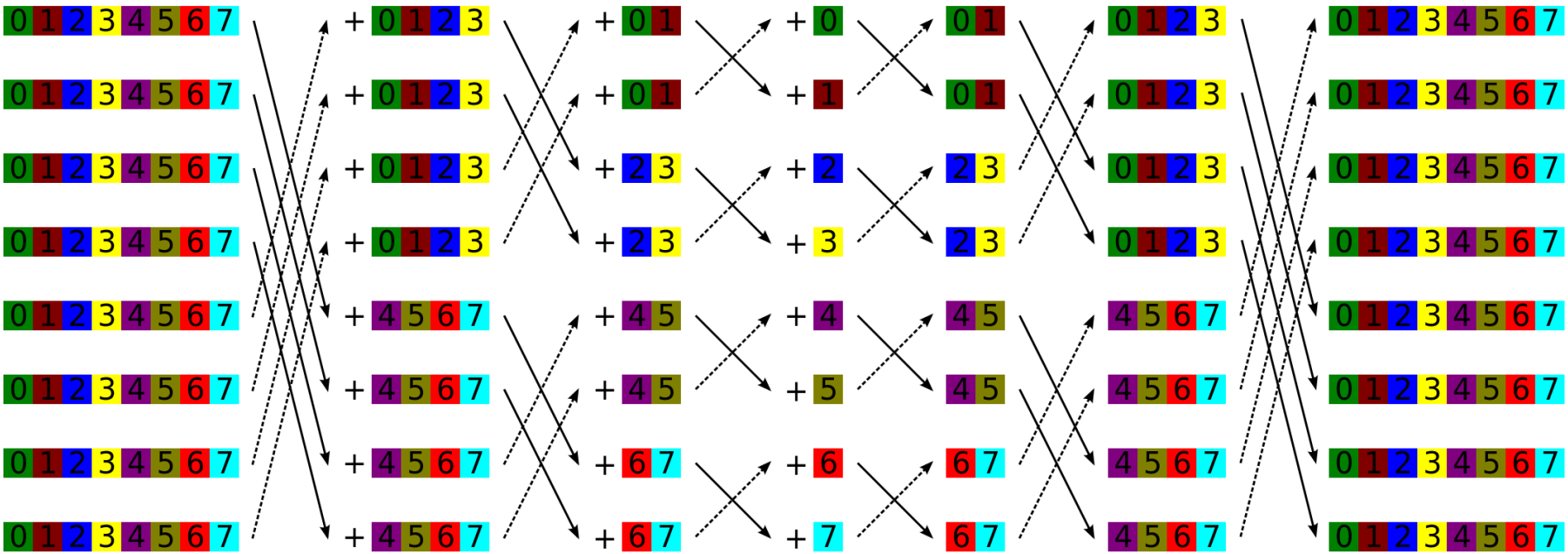
- no pipelining necessary to achieve linear bandwidth cost

Butterfly Reduce-Scatter (recursive halving)

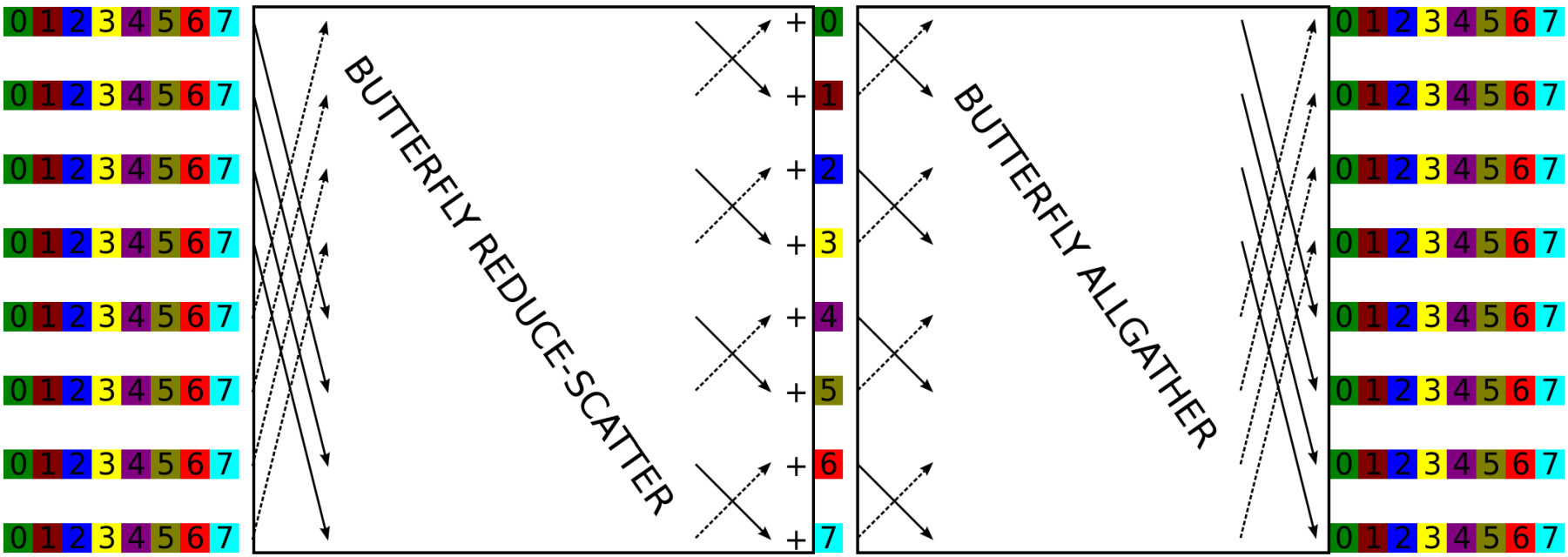


$$T_{\text{reduce-scatter}} = T_{\text{allgather}} + \text{cost of local reduction work}$$

Butterfly Allreduce

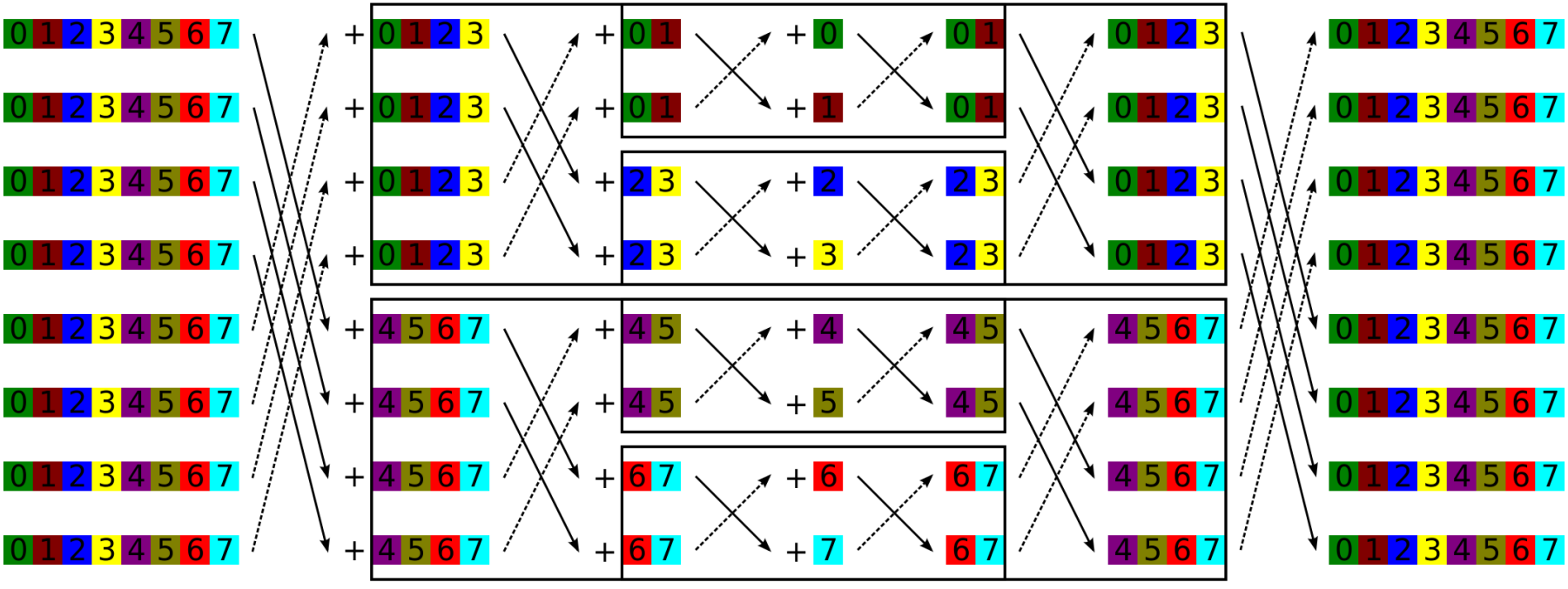


Butterfly Allreduce



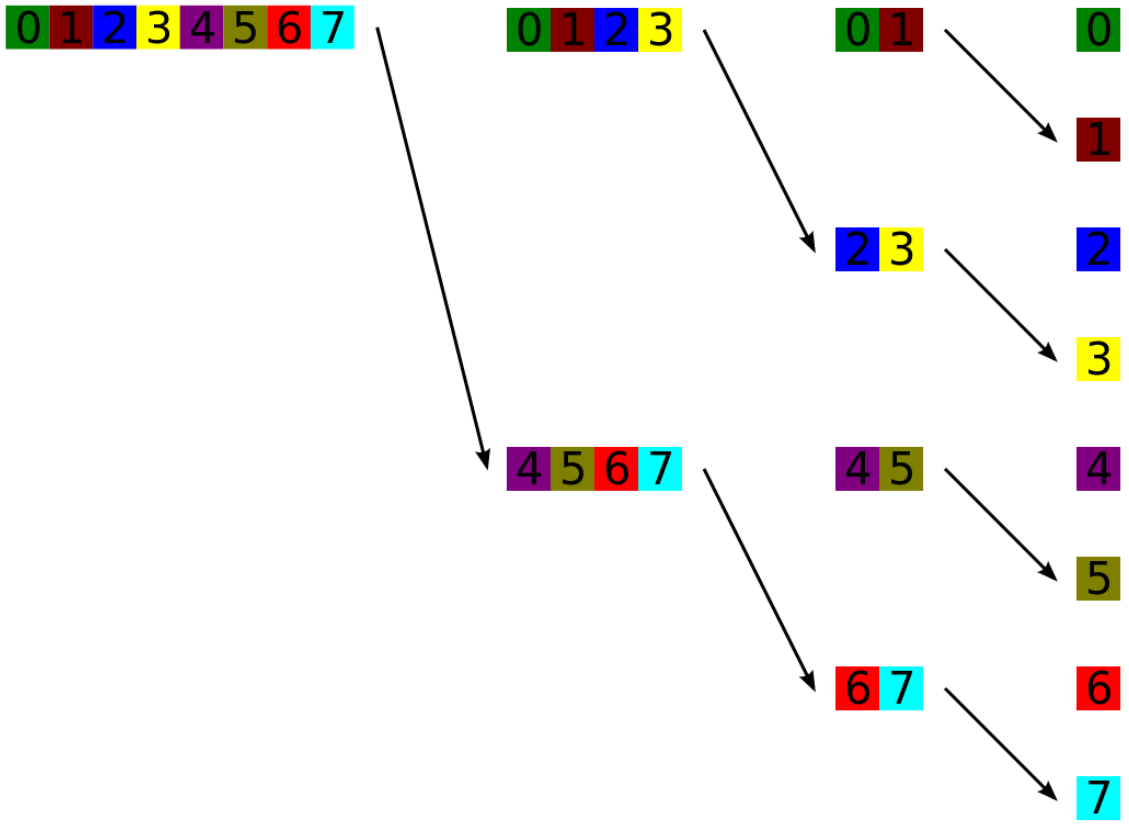
$$T_{\text{allreduce}} = T_{\text{reduce-scatter}} + T_{\text{allgather}}$$

Butterfly Allreduce: note recursive structure of butterfly



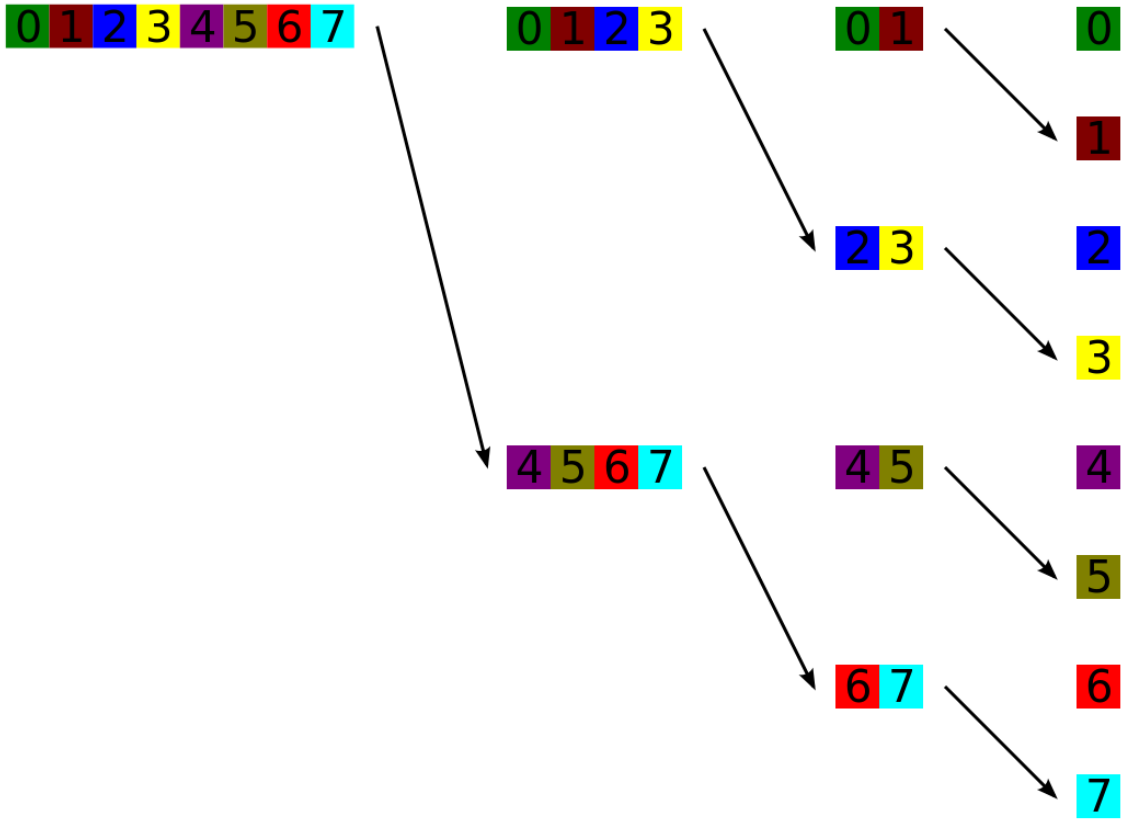
Its possible to do Scan (each processor ends up with a unique value of a prefix sum rather than the full sum) in a similar fashion, but also with operator application done additionally during recursive doubling (Allgather)

Butterfly Scatter



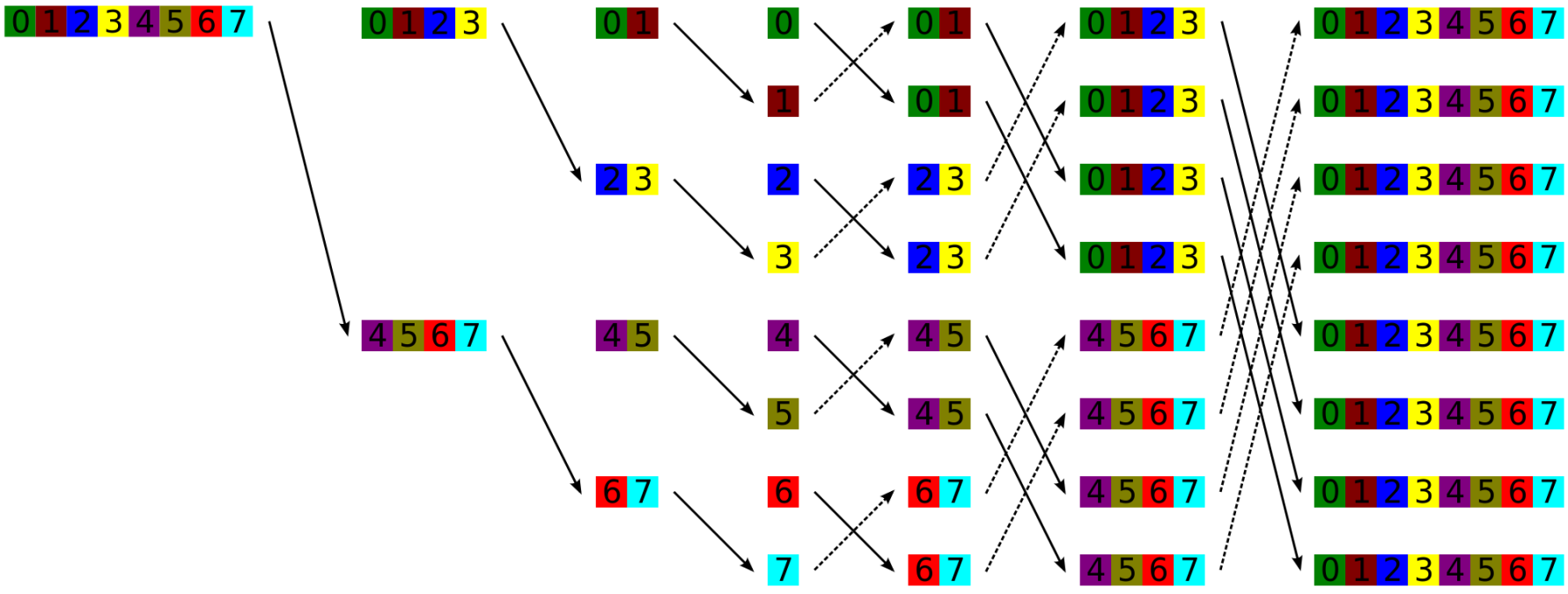
Question: Which tree is this equivalent to?

Butterfly Scatter

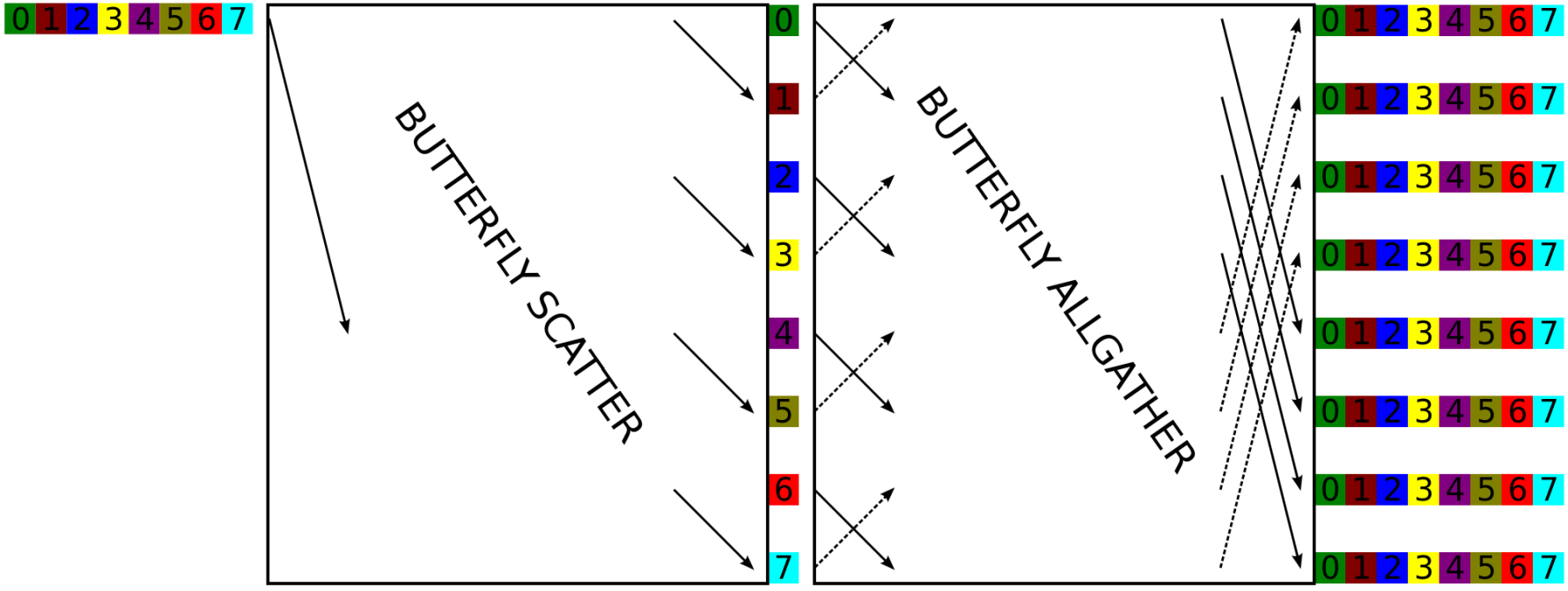


Question: Which tree is this equivalent to?
Answer: Binomial tree.

Butterfly Broadcast

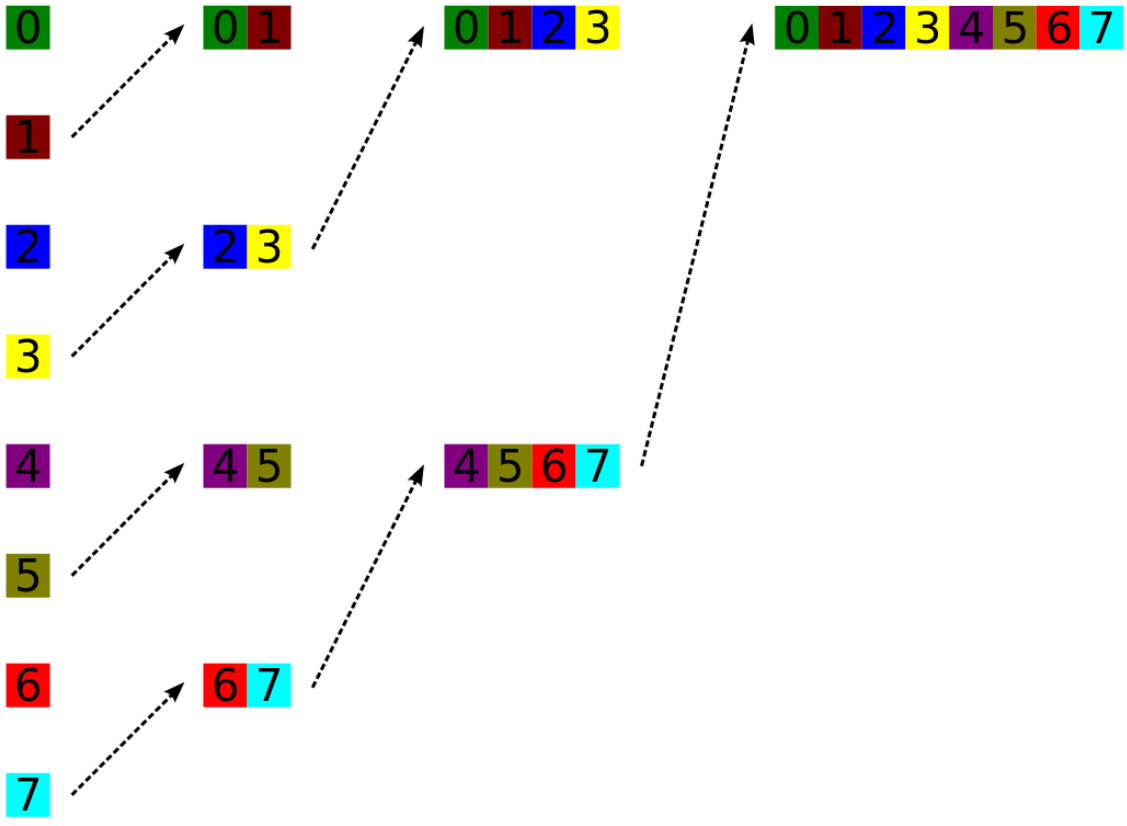


Butterfly Broadcast



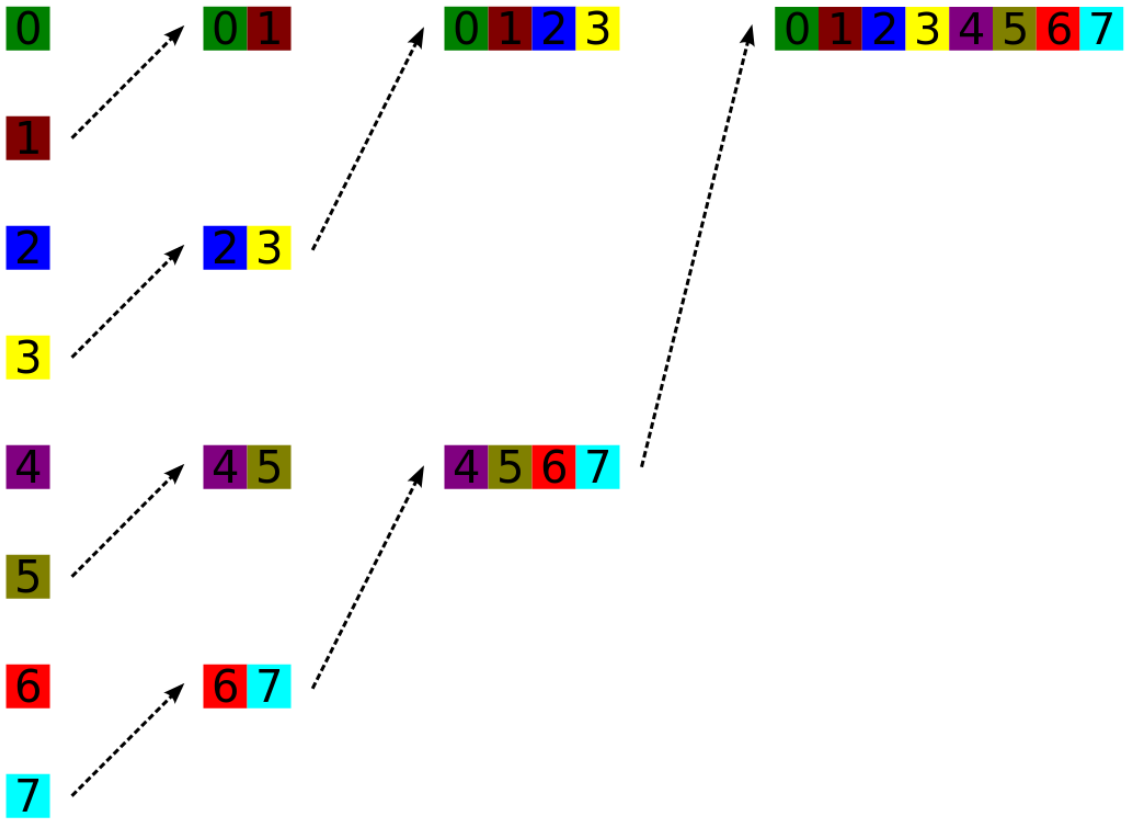
$$T_{\text{broadcast}} = T_{\text{scatter}} + T_{\text{allgather}}$$

Butterfly Gather



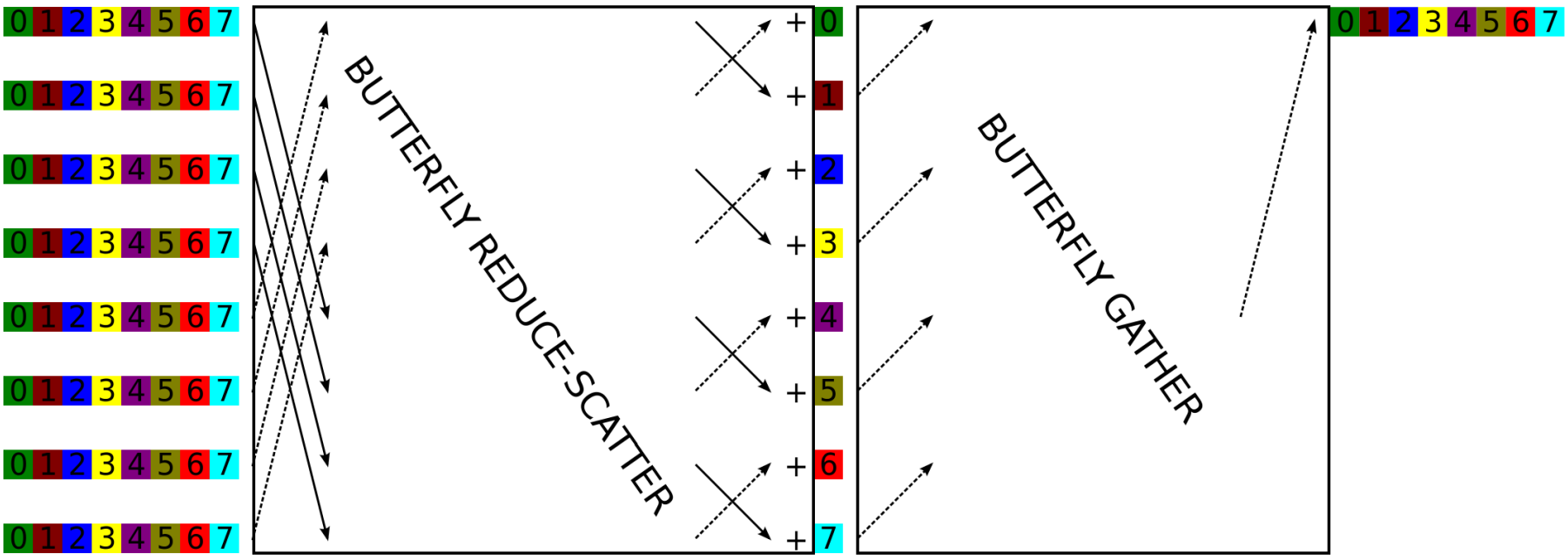
Question: Which other collective could use Gather as a subroutine?

Butterfly Gather



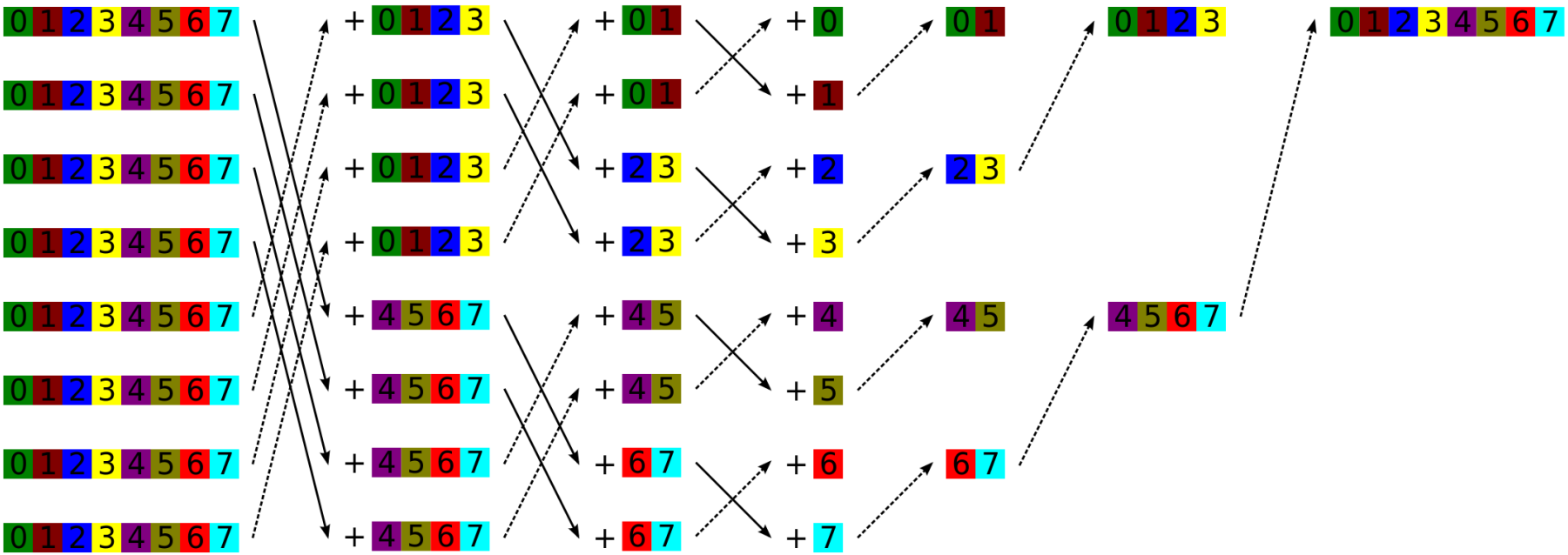
Question: Which other collective could use Gather as a subroutine?
Answer: Reduction.

Butterfly Reduce



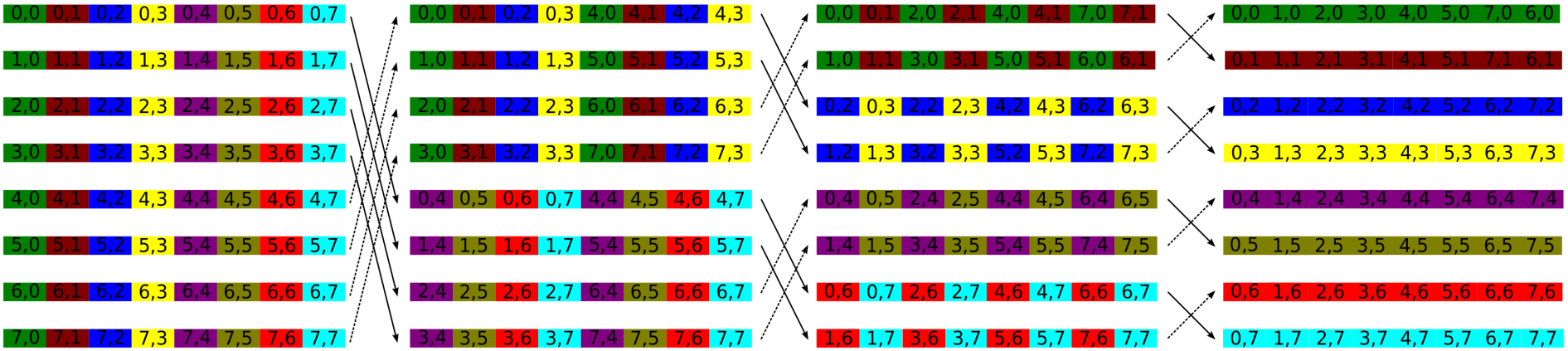
$$T_{\text{reduce}} = T_{\text{reduce-scatter}} + T_{\text{gather}}$$

Butterfly Reduce



$$T_{\text{reduce}} = T_{\text{reduce-scatter}} + T_{\text{gather}}$$

Butterfly All-to-All



Note that the size of the message stays the same at each level

$$T^{\alpha,\beta}_{\text{all-to-all}}(s,P) = \begin{cases} 0 & : P = 1 \\ T^{\alpha,\beta}_{\text{all-to-all}}(s,P/2) + \alpha + (s/2)\cdot\beta & : P > 1 \end{cases}$$

$$= \alpha \cdot \log_2(P) + \beta \cdot \sum_{i=1}^{\log_2(P)} s/2 = \alpha \cdot \log_2(P) + \beta \cdot s/2 \cdot \log_2(P)$$

Its possible to do All-to-All in less bandwidth cost (as low as $\beta \cdot s$ by sending directly to targets) at the cost of more messages (as high as $\alpha \cdot P$ if sending directly)