コンピュータ科学特別講義Ⅳ

Parallel Algorithm Design（#3）

Masato Edahiro

May 25, 2018

Class WWW: http://www.pdsl.jp/class/utyo2018/

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.

PDSL

Contents of This Class

- Our Target
  - Understand Systems and Algorithms on “Multi-Core” processors
- Schedule (Tentative)
  - #1 April 6 (=Today) What’s “Multi-Core”?
  - #2 April 13 : Parallel Programming Languages (Ex. 1)
  - April 20, 27, May 4, 11, 18: NO CLASS
  - #3 May 25 : Parallel Algorithm Design
  - #4 June 1 (Fri) : Laws on Multi-Core
  - #5 June 8 : Examples of Parallel Algorithms (1) (Ex. 2)
  - June 15: NO CLASS
  - #6 June 22 : Examples of Parallel Algorithms (2)
  - #7 June 29 : Examples of Parallel Algorithms (3)
  - #8 July 6 : Examples of Parallel Algorithms (4)
  - #9 July 13 : Examples of Parallel Algorithms (5) (Ex. 3)
  - (July 20)

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.
PDSL

並列アルゴリズム設計

- 今日のトピック
  - Fosterによる設計方法論
  - Example: Reduction処理の並列化
  - 参考文献:
    M. Quinn: Parallel Programming in C with MPI and OpenMP, McGraw-Hill, 2003.
    (This book refers: I. Foster: Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering, Addison Wesley, 1995.)
    http://www-unix.mcs.anl.gov/dbpp

並列アルゴリズム設計

- タスク・チャネルモデル
  - タスク: プログラム、内蔵メモリ、入出力ポート
  - チャネル：メッセージキュー。あるタスクの出力ポートと別のタスクの入力ポートを接続する
  - 受信タスク：ブロック、同期：タスクは出力タスクからの値が準備できるまで先に進めない
  - 送信タスク: 非ブロック、非同期：タスクは入力タスクの受信準備を待たずに先に進める

Task / Channel Model

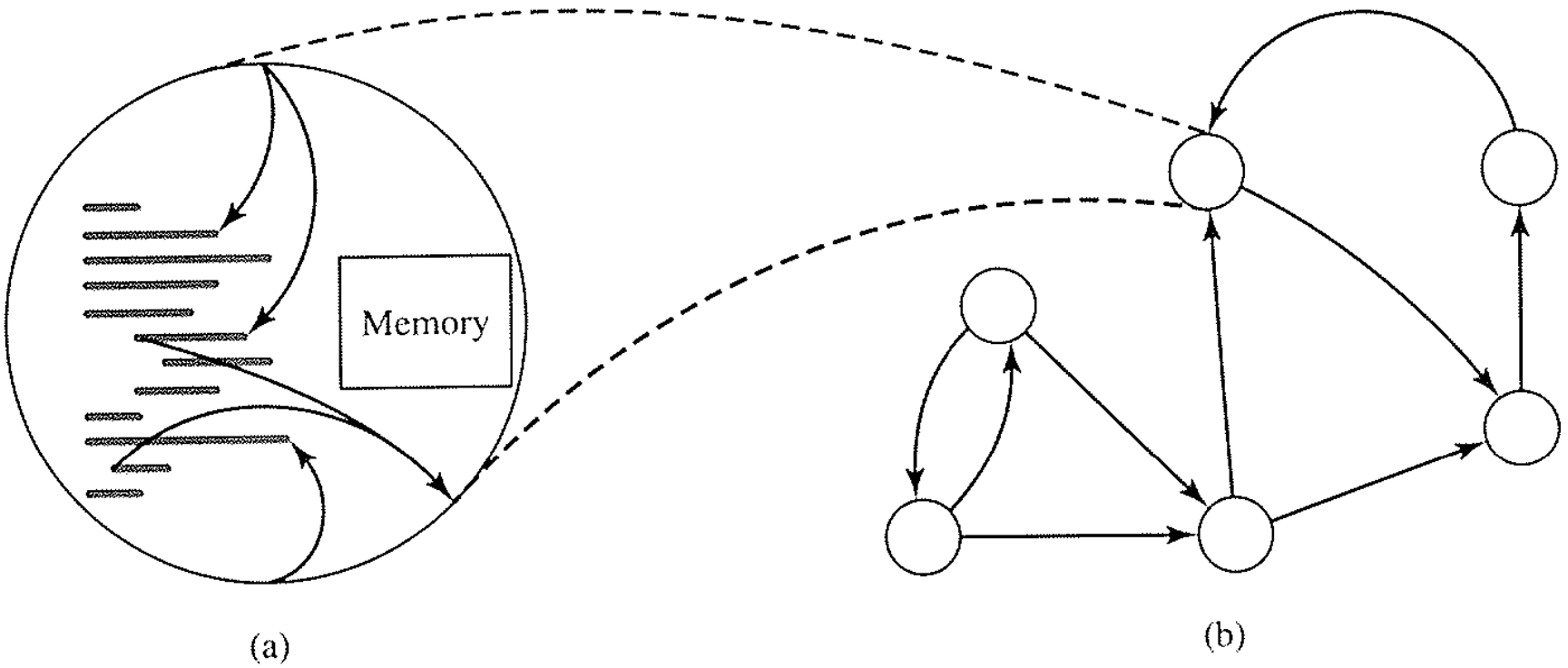


Figure 3.1 The task/channel programming model. (a) A task consists of a program, local memory, and a collection of I/O ports. (b) A parallel computation can be viewed as a directed graph in which vertices represent tasks and directed edges represent communication channels.

Fosterによる設計方法論

（まずは、プロファイラなどを使ってホットスポット（＝もっとも時間がかかっていて並列化する価値がある関数など）を見つける）

1. Partitioning
2. Communication
3. Agglomeration
4. Mapping

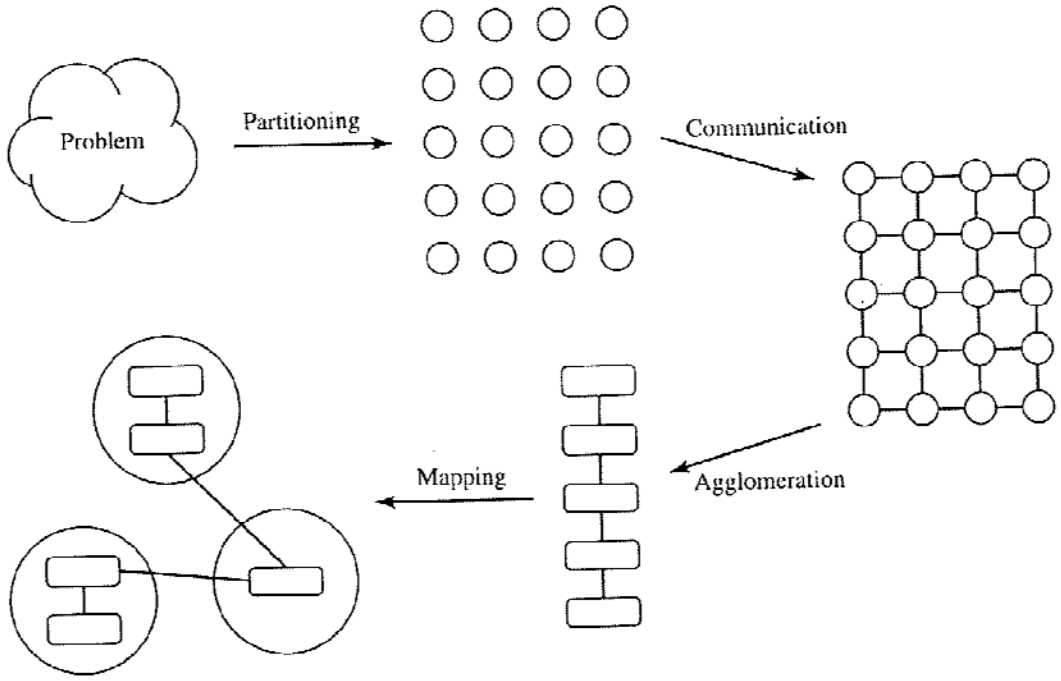


Figure 3.2 Foster's parallel algorithm design methodology.

Partitioning

- 可能な限りの並列性を見つけ、要素タスクを抽出する
  - Domain decomposition
    1. データを分割する
    2. 分割されたデータに計算を関連付ける方法を決める
  - Functional Decomposition
    1. 計算を分割する
    2. 分割された個々の計算にデータを関連付ける方法を決める

    - しばしばパイプライン手法が使われる

Domain Decomposition

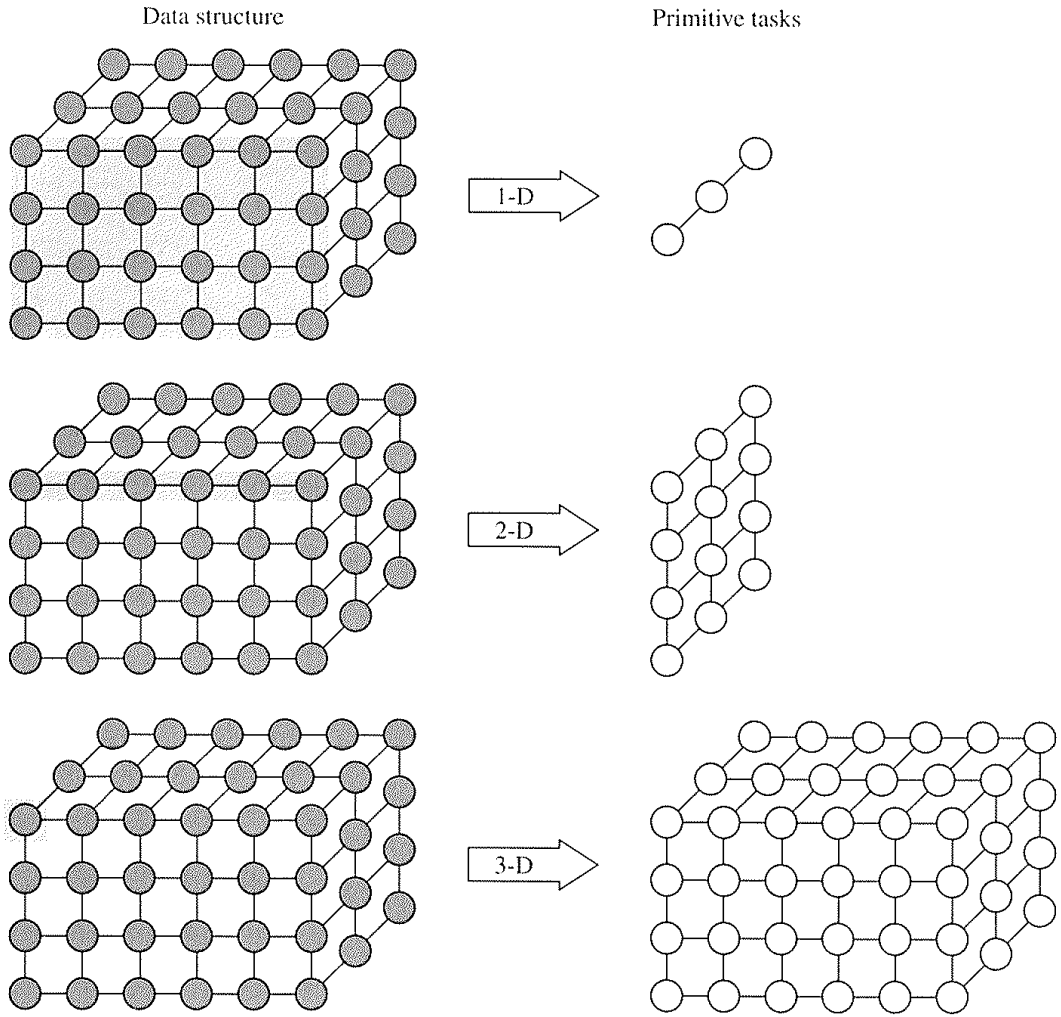
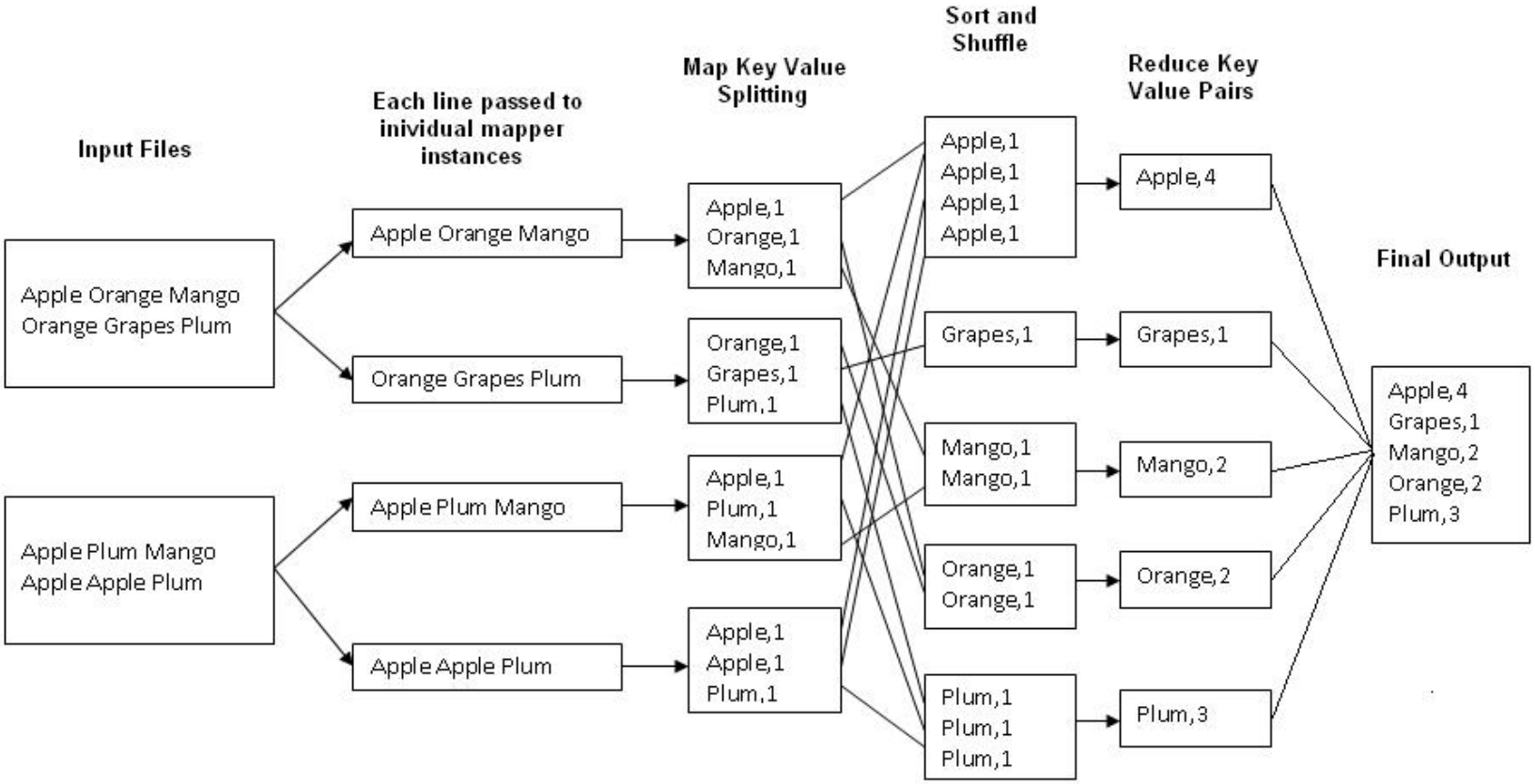


Figure 3.3 Three domain decompositions of a three-dimensional matrix, resulting in markedly different collections of primitive tasks.

Map-Reduce


Input Files
Each line passed to inividual mapper instances
Map Key Value Splitting
Sort and Shuffle
Reduce Key Value Pairs
Final Output
Apple Orange Mango Orange Grapes Plum
Apple Plum Mango Apple Apple Plum
Apple Orange Mango
Orange Grapes Plum
Apple Plum Mango
Apple Apple Plum
Apple,1 Orange,1 Mango,1
Orange,1 Grapes,1 Plum,1
Apple,1 Plum,1 Mango,1
Apple,1 Apple,1 Plum,1
Apple,1 Apple,1 Apple,1 Apple,1
Grapes,1
Mango,1 Mango,1
Orange,1 Orange,1
Plum,1 Plum,1 Plum,1
Apple,4
Grapes,1
Mango,2
Orange,2
Plum,3
Apple,4 Grapes,1 Mango,2 Orange,2 Plum,3

Functional Decomposition

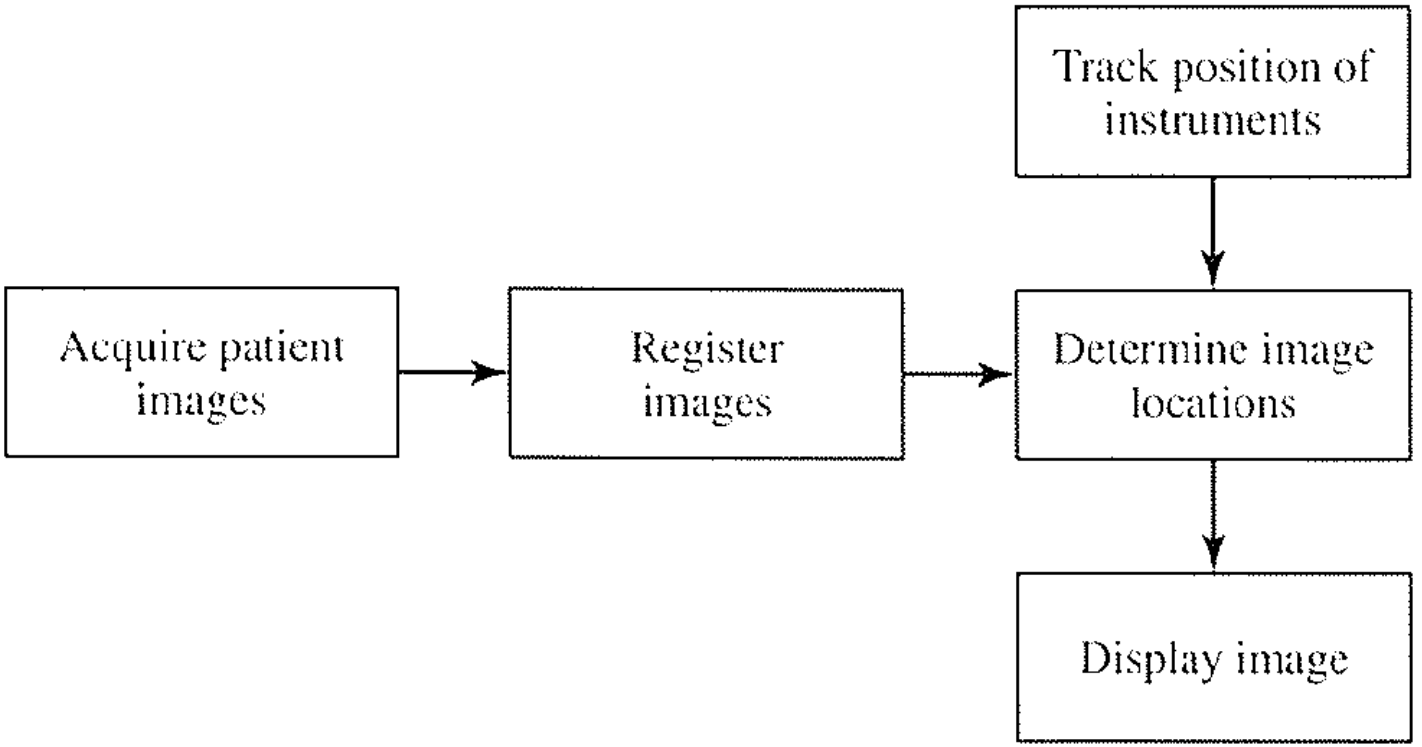
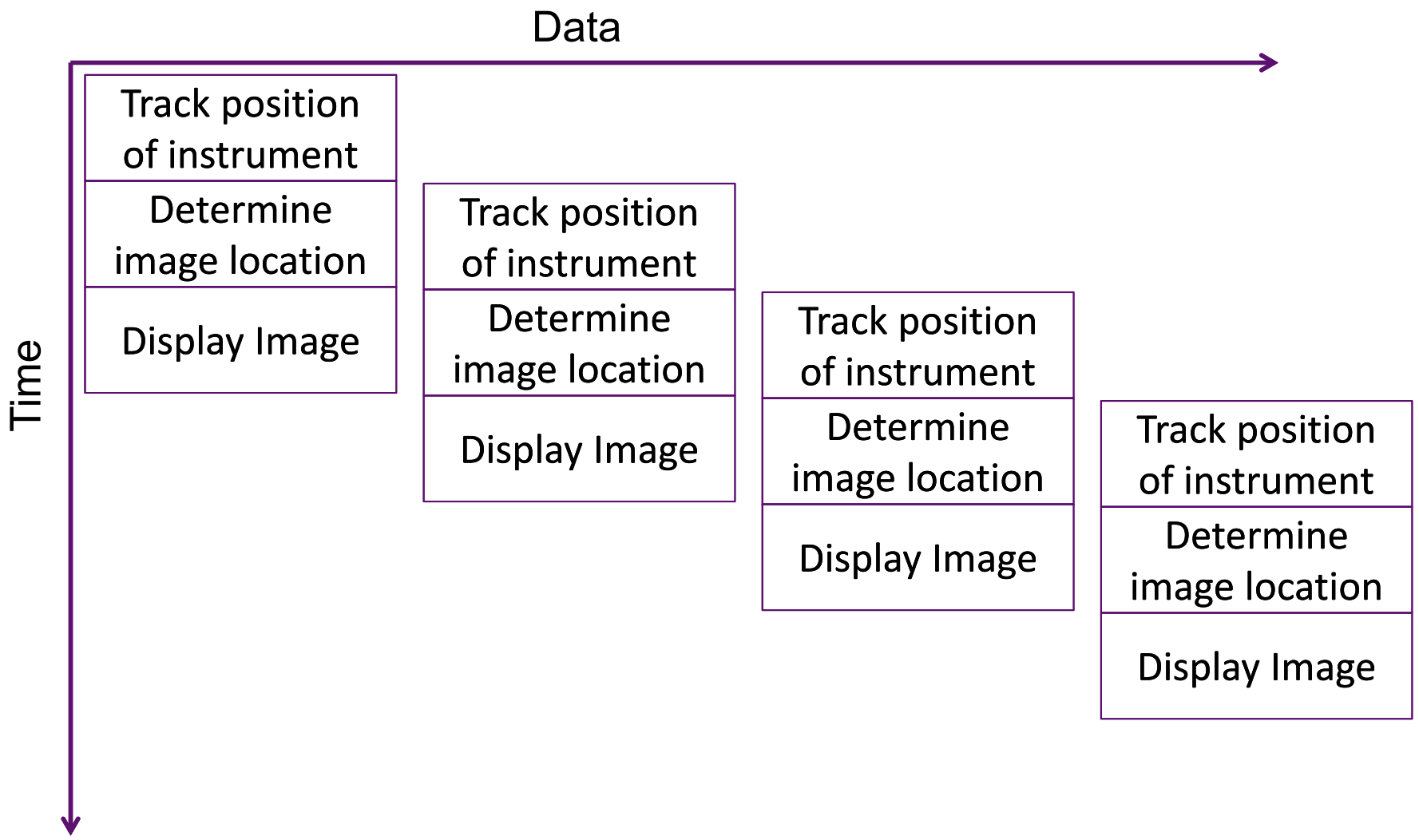


Figure 3.4 Functional decomposition of a system supporting interactive image-guided surgery.

Pipelining


Data
Time
Track position of instrument
Determine image location
Display Image
Track position of instrument
Determine image location
Display Image
Track position of instrument
Determine image location
Display Image
Track position of instrument
Determine image location
Display Image

Fosterによるチェックリスト (Partitioning)

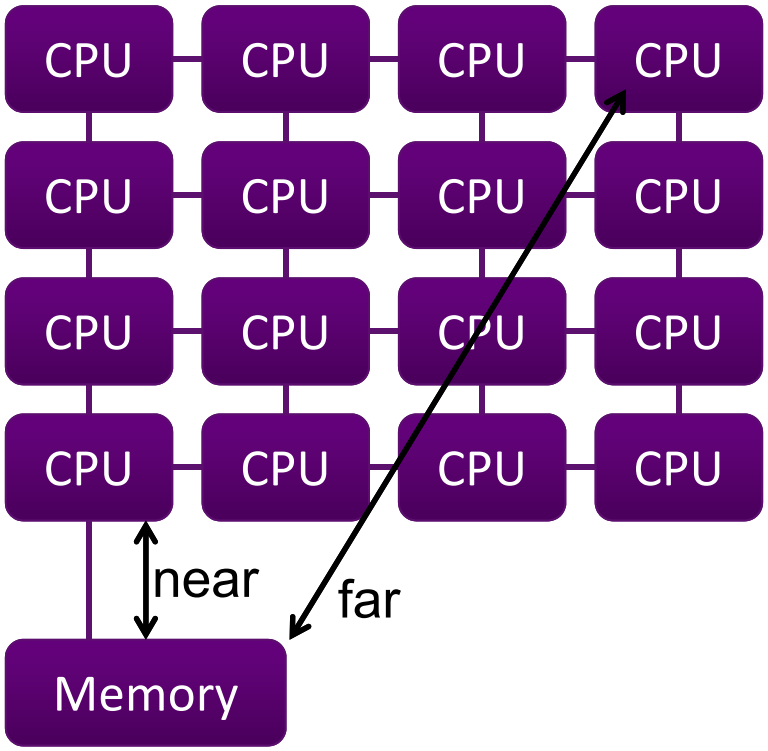
- 少なくとも想定する計算機が持つプロセッサ数よりも一桁多いタスクがあること (この条件が満たされない場合、後の設計自由度が大幅に制約を受ける)
- 冗長な計算やデータ構造が最小化されていること (この条件が満たされない場合、問題規模が大きくなった時に十分な性能が得られない場合がある)
- 要素タスクはおおよそ同じサイズであること (もしも違うならば、プロセッサ間の負荷分散が難しくなる場合がある)
- タスク数が問題規模に応じた増加関数であること (もしも違うならば、より規模の大きい問題に対して、より多くのプロセッサを使うことが不可能になる場合がある)

Communication

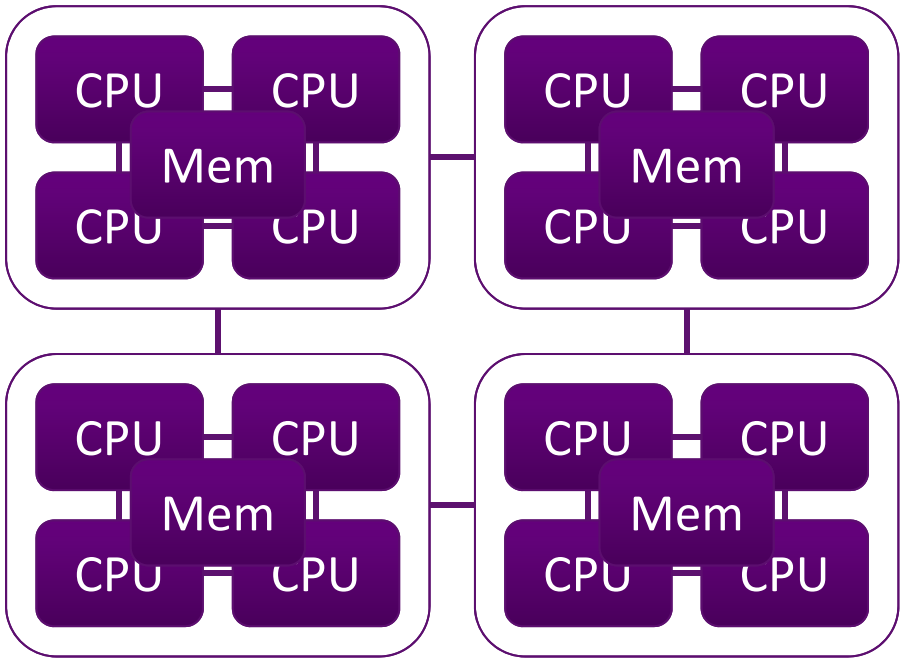
- 局所的通信
  - ある計算実行において少数のタスクのみが関与する
- 大域的通信
  - ある計算実行において多くのタスクが関与する

- 大域的な通信は局所的通信と比べてオーバーヘッドが大きい
  - アーキテクチャに依存する
  - 通信に分散メモリ型を使う場合大きなオーバーヘッドとなる

Examples of Many-Core Architecture

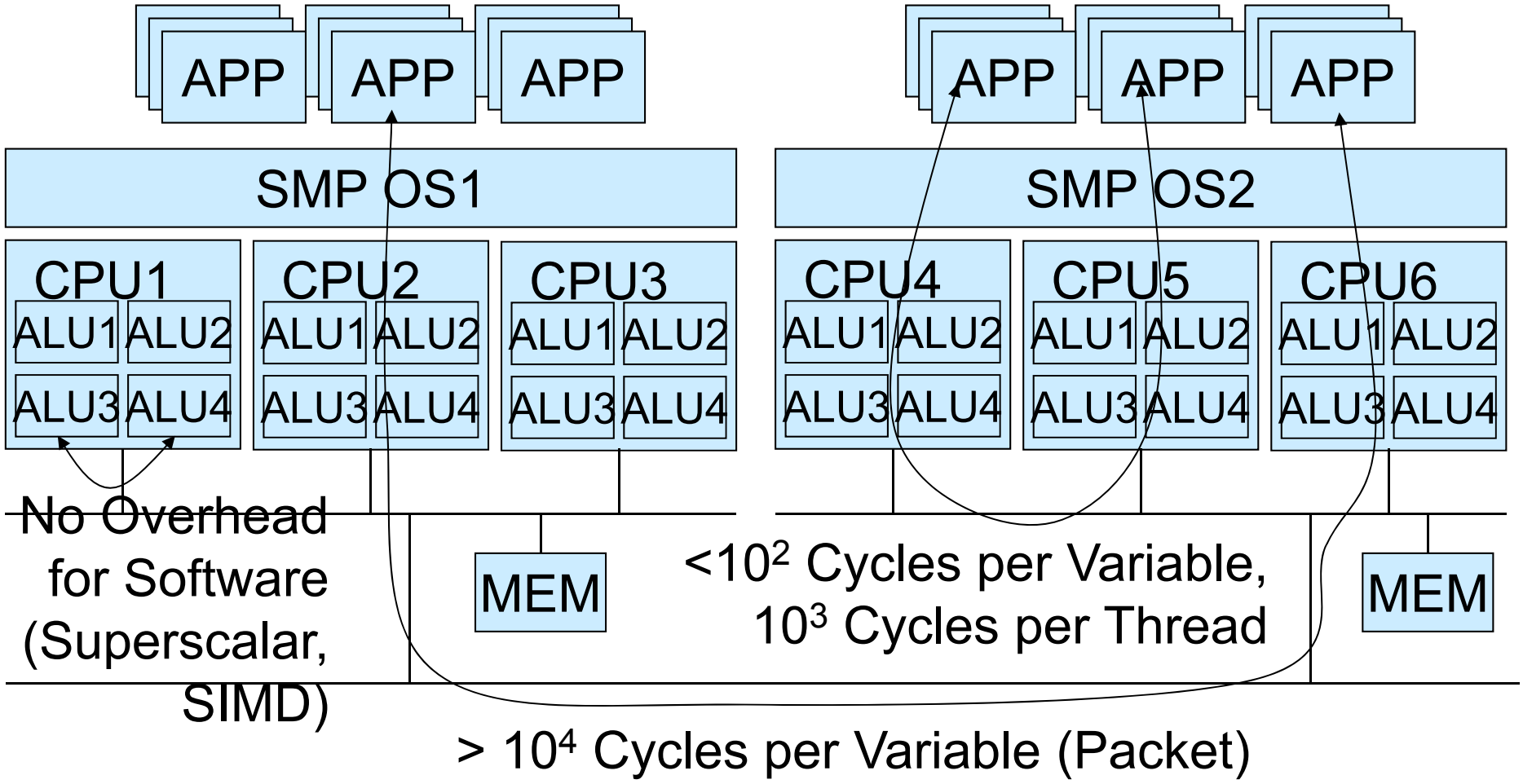


Cf. Tileraは等距離



クラスタ内は集中メモリ
クラスタ間は分散メモリ

通信時間


APP
APP
APP
APP
APP
APP
SMP OS1
SMP OS2
CPU1
CPU2
CPU3
CPU4
CPU5
CPU6
ALU1
ALU2
ALU3
ALU4
No Overhead for Software (Superscalar, SIMD)
MEM
MEM
<10^2 Cycles per Variable,
10^3 Cycles per Thread
> 10^4 Cycles per Variable (Packet)

Fosterによるチェックリスト (Communication)

- 通信はタスク間でバランスされていること
- 各タスクは少数の隣接タスクとのみ通信していること
- それぞれのタスクが並行して通信処理を実行できること
- それぞれのタスクが並行して計算処理を実行できること

Agglomeration（集塊、凝集）

- 性能やプログラム容易性を向上させるため、タスクのグループ化を行ってより大きなタスクにする処理
  - 通信オーバーヘッドの減少
    - 局所性の増大
    - 送信タスクや受信タスクのグループ化
  - 並列設計のスケーラビリティの維持
    - 例：(8 x 128 x 256) 3 次元行列演算
    - 第一次元でagglomerationを行うと、
      - 4 CPU: 2 x 128 x 256 per CPU
      - 8 CPU: 1 x 128 x 256 per CPU
      - 8CPU以上になると設計変更が必要

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.
PDSL

通信オーバーヘッドの減少

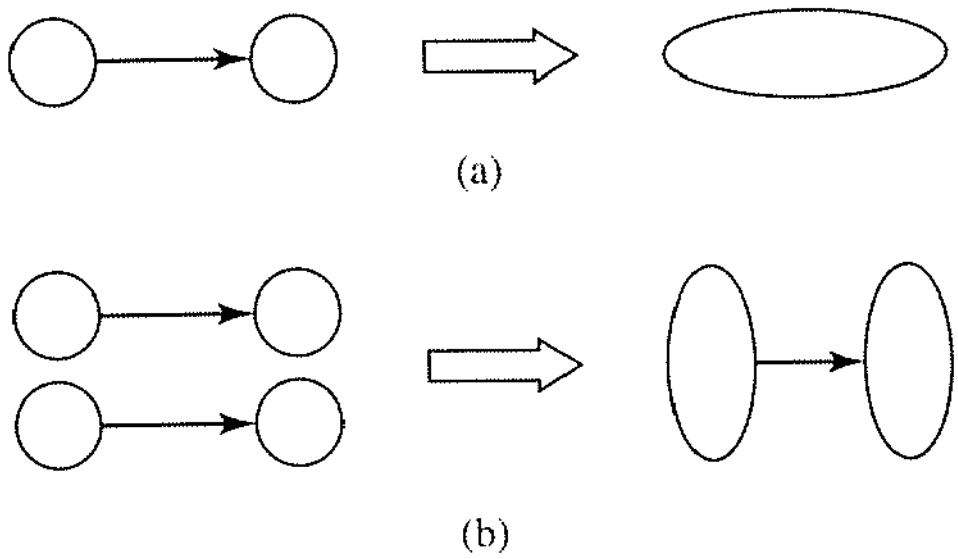
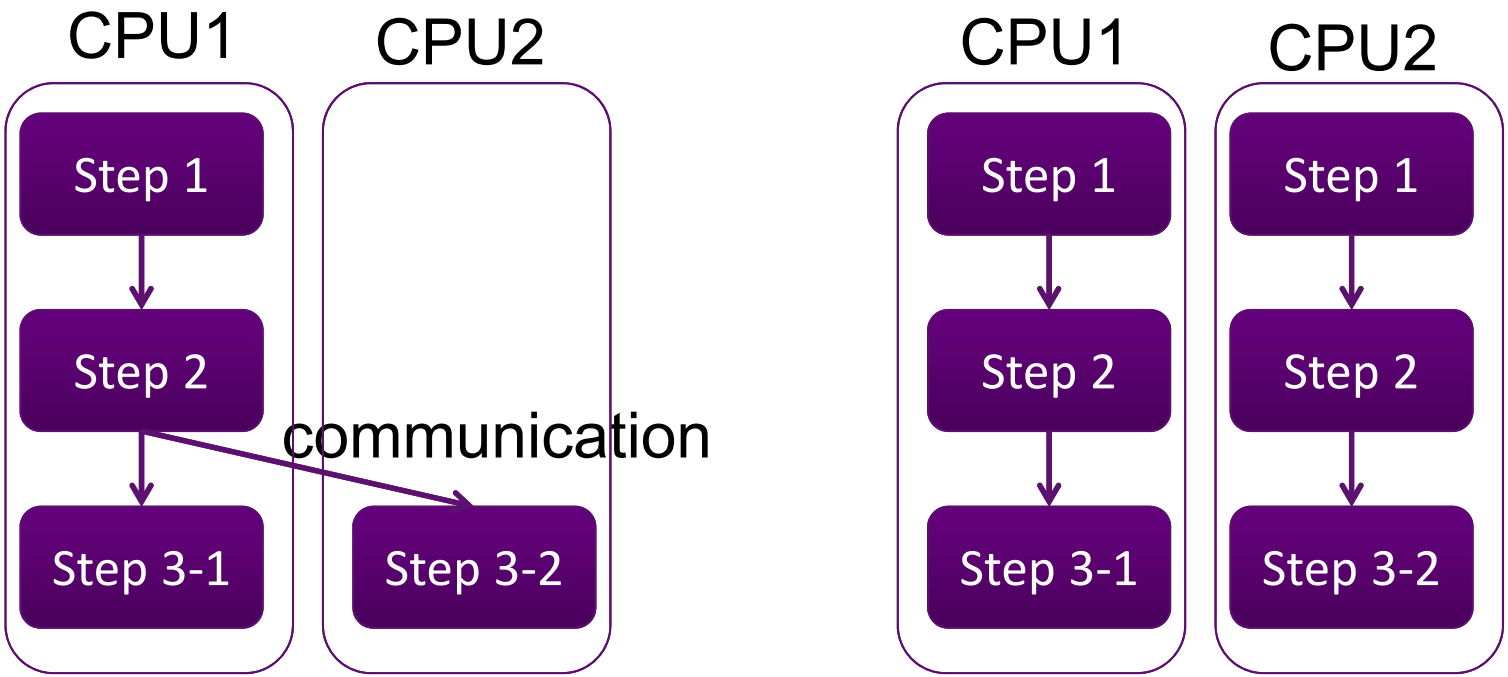


Figure 3.5 Agglomerating tasks can eliminate communications or at least reduce their overhead. (a) Combining tasks that are connected by a channel eliminates that communication, increasing the locality of the parallel algorithm. (b) Combining sending and receiving tasks reduces the number of message transmissions.

Fosterによるチェックリスト (Agglomeration)

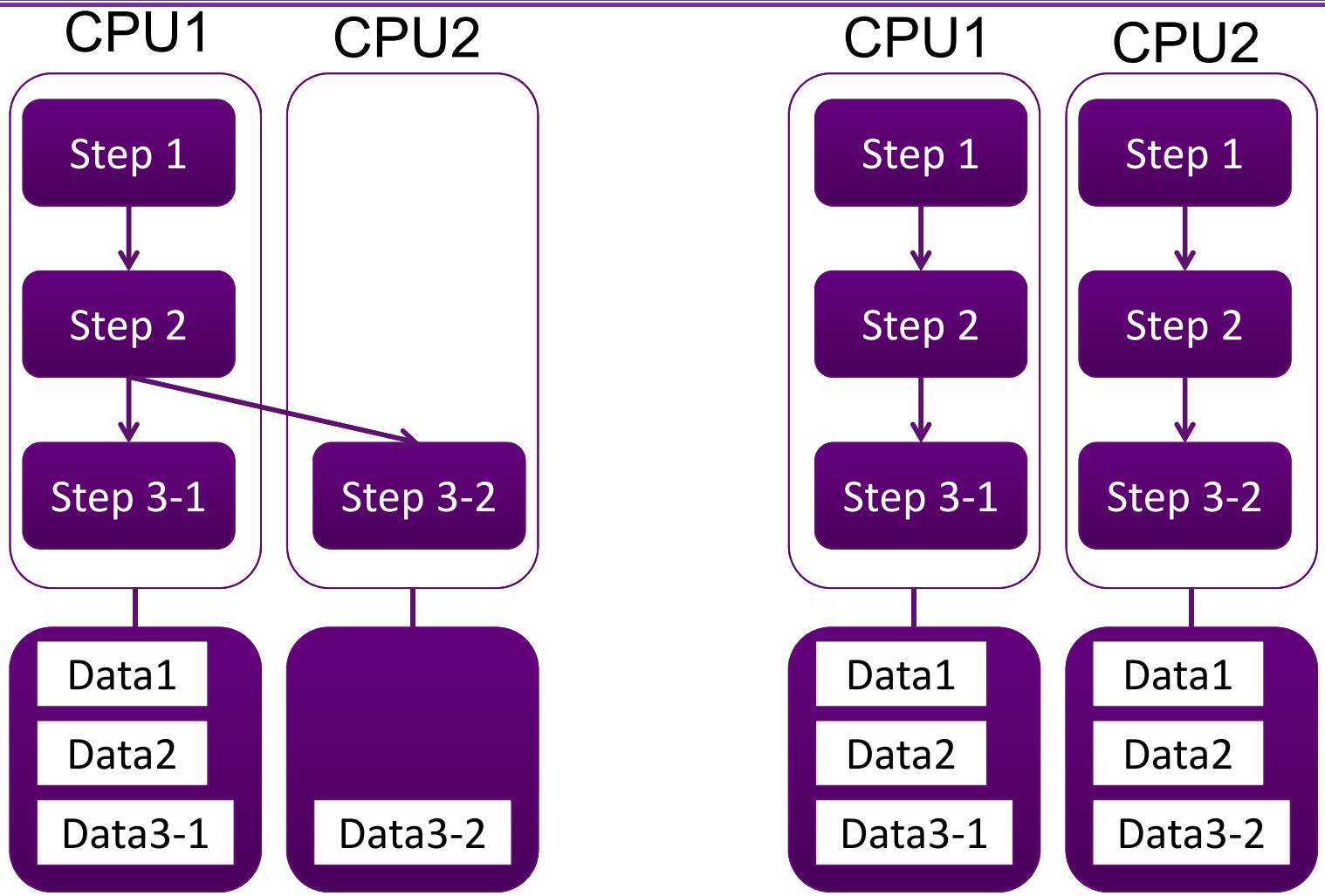
- Agglomerationによって並列アルゴリズムの局所性が増大していること
- 計算の複製によって通信を利用するよりも処理時間が減少していること
- データの複製は十分少なく、アルゴリズムのスケーラビリティを損なっていないこと
- グループ化後のそれぞれタスクは同様の計算および通信時間を有していること
- タスク数は問題規模に応じて増加関数であること
- タスク数は可能な限り少なく、しかしながら少なくとも想定計算機のプロセッサ数と同程度であること
- Agglomerationの選択と既存コード改変コストとのトレードオフが妥当であること

計算の複製



どちらが高速?

データの複製



どちらが高速?

Mapping

- タスクをプロセスに割り当てる処理
- 分散メモリを仮定
  - 集中メモリ型計算機であれば、ＯＳが自動的に割り当てるため
- プロセッサ利用率最大化かつプロセッサ間通信最小化
  - プロセッサ利用率 --- 現在考えている問題の解法において実際にプロセッサが動作している時間の割合の平均値

Mappingの例

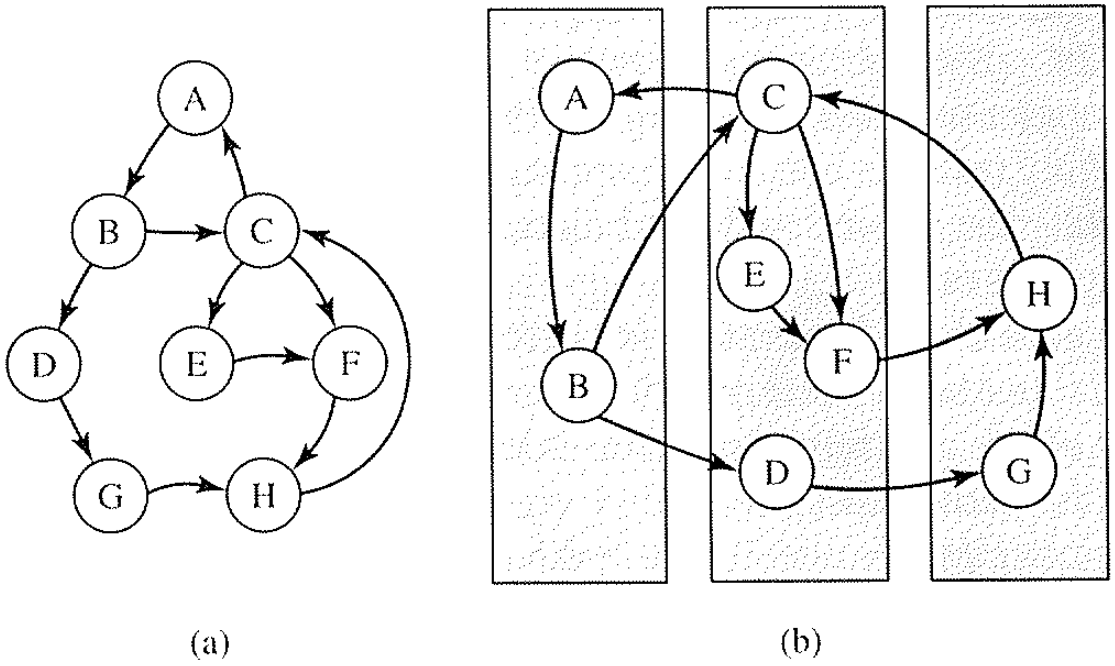


Figure 3.6 The mapping process. (a) A task/channel graph. (b) Mapping of tasks to three processors. Some channels now represent intraprocessor communications, while others represent interprocessor communications.

真ん中のプロセッサはタスクもプロセッサ間通信も多い

Mapping最適化

- プロセッサ利用率最大化とプロセッサ間通信最小化はしばしば競合する.
- 例：すべてのタスクを一つのプロセッサに割り付ければ
  → プロセッサ利用率は最悪であるが、プロセッサ間通信は最善（０）となる
- NP-Hard

Mapping手法

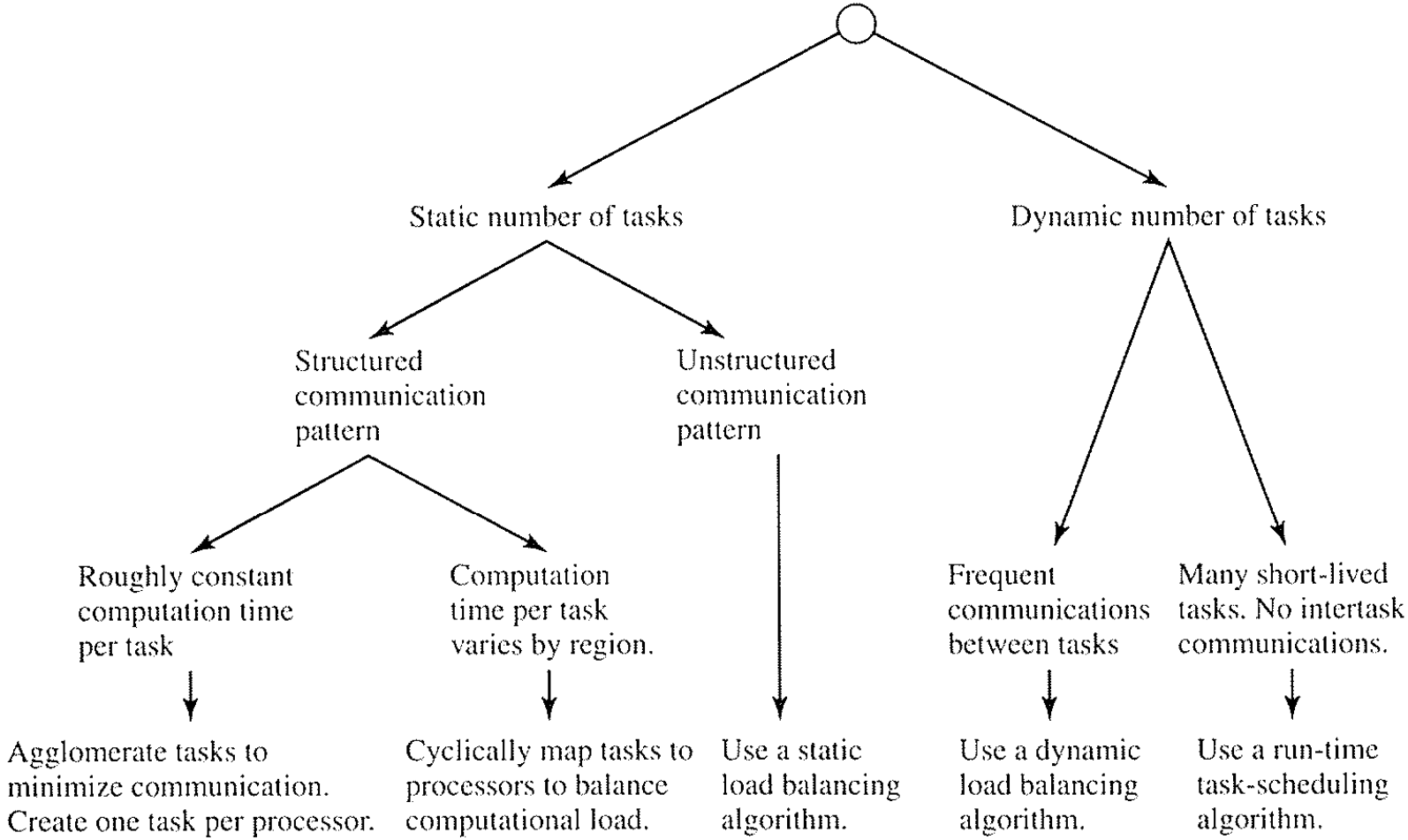


Figure 3.7 A decision tree to choose a mapping strategy. The best strategy depends on characteristics of the tasks produced as a result of the partitioning, communication, and agglomeration steps.

Fosterによるチェックリスト (Mapping)

- プロセッサあたり１タスクにする設計と、プロセッサあたり複数タスクにする設計について考慮済であること
- 静的および動的割付について評価済であること
- タスクの動的割付を使う場合、管理プログラム（タスクアロケータ）が性能のボトルネックになっていないこと
- タスクの静的割当を使う場合、タスク数とプロセッサ数の割合が少なくとも10:1はあること

並列アルゴリズム設計

- 今日のトピック
  - Fosterによる設計方法論
  - Example: Reduction処理の並列化
  - 参考文献:
    M. Quinn: Parallel Programming in C with MPI and OpenMP, McGraw-Hill, 2003.
    (This book refers: I. Foster: Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering, Addison Wesley, 1995.)

Reduction演算の並列化

- Reduction
  - n個の変数$a_0, a_1, \cdots, a_{n-1}$と結合則を満たす２項演算⊕に対し、$a_0 \oplus a_1 \oplus \cdots \oplus a_{n-1}$の計算処理をreductionとよぶ
  - 加算、最大、最小など
  - 逐次計算処理においてちょうどn-1回の演算であるため時間複雑度はΘ(n)
    並列計算によりどれだけ実行時間を高速化できるか？
  - 以下では一般性を損なうことなく加算(+)を用いて説明する

Partitioning & Communication

- N個の要素タスクに分割可能
- λ: 値一つあたりのタスク間通信時間
- χ: 加算に必要な実行時間
- 実行時間は？
- 実行時間を最小化するには？

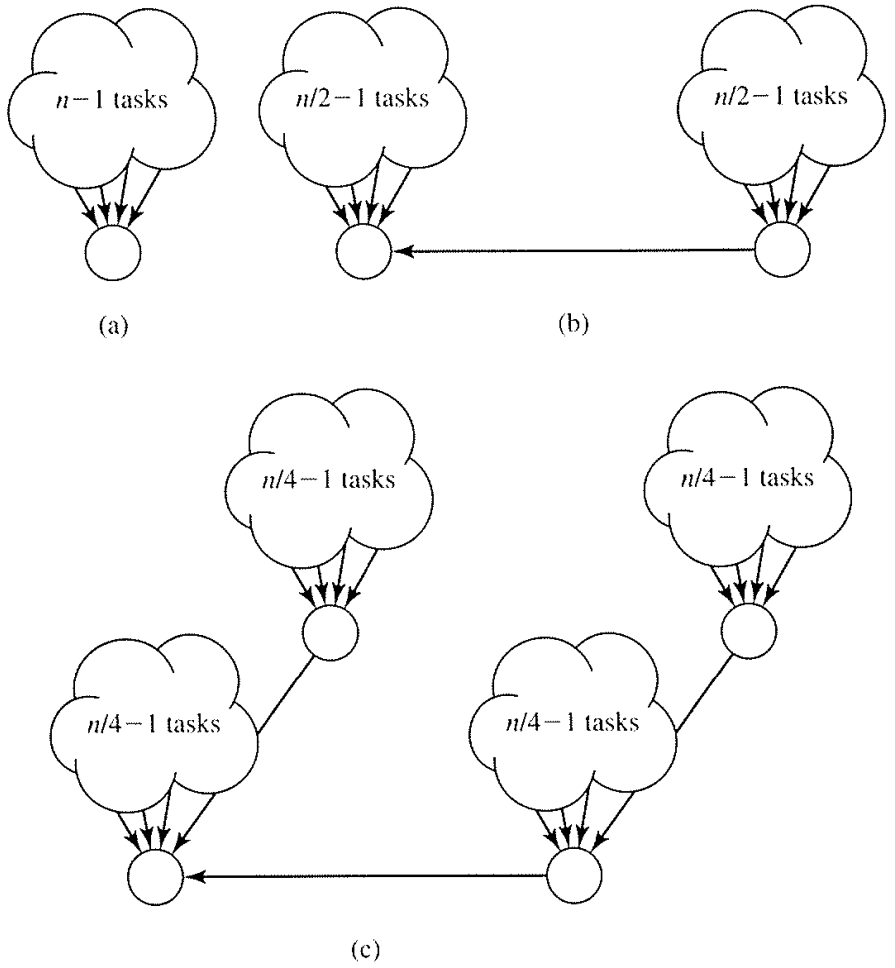


Figure 3.12 Evolution of an efficient parallel algorithm for reduction. (a) One task receives a list element from each of the other $n - 1$ tasks and performs all the additions. (b) Two tasks work together. Each receives list elements from $n/2 - 1$ other tasks. After $n/2$ addition steps, one task sends its subtotal to the first task. Compared to the original version, the computation time is cut nearly in half. (c) Four tasks cooperate. Each receives list elements from $n/4 - 1$ other tasks. After $n/4$ concurrent addition steps, there are four subtotals. These can be combined in two more communication/computation steps.

Binomial trees

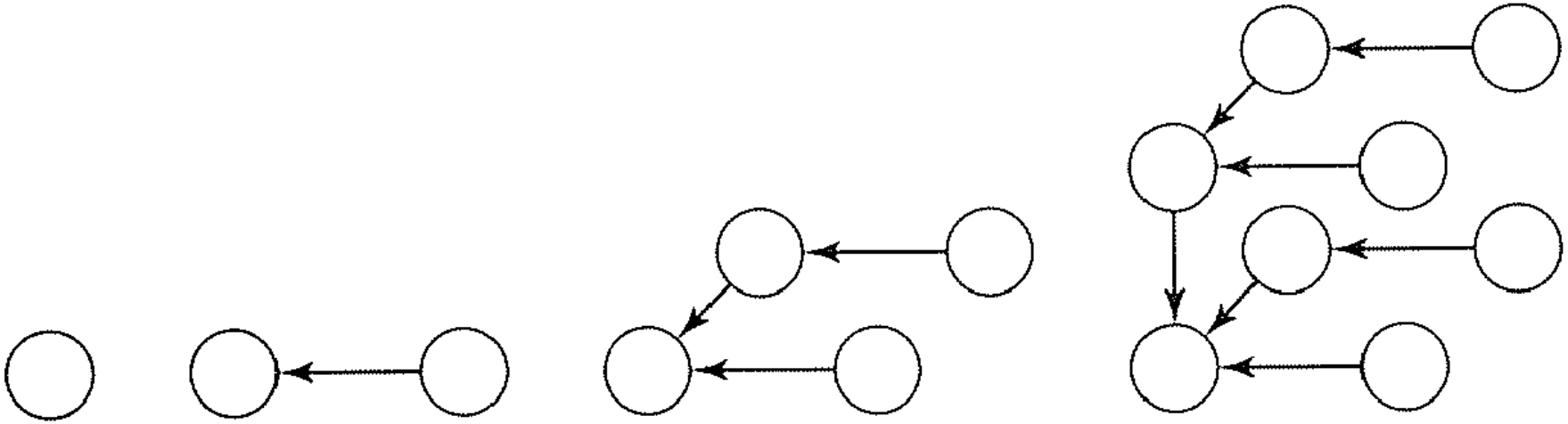


Figure 3.13 Binomial trees with 1, 2, 4, and 8 nodes.

Binomial trees

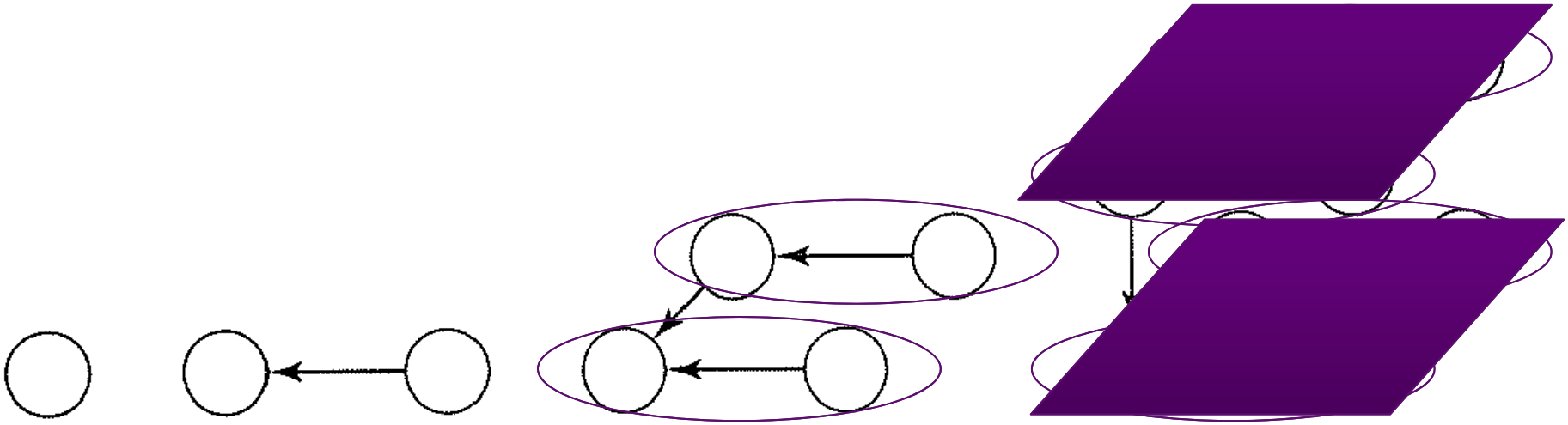


Figure 3.13 Binomial trees with 1, 2, 4, and 8 nodes.

Finding the global sum in logarithmic time

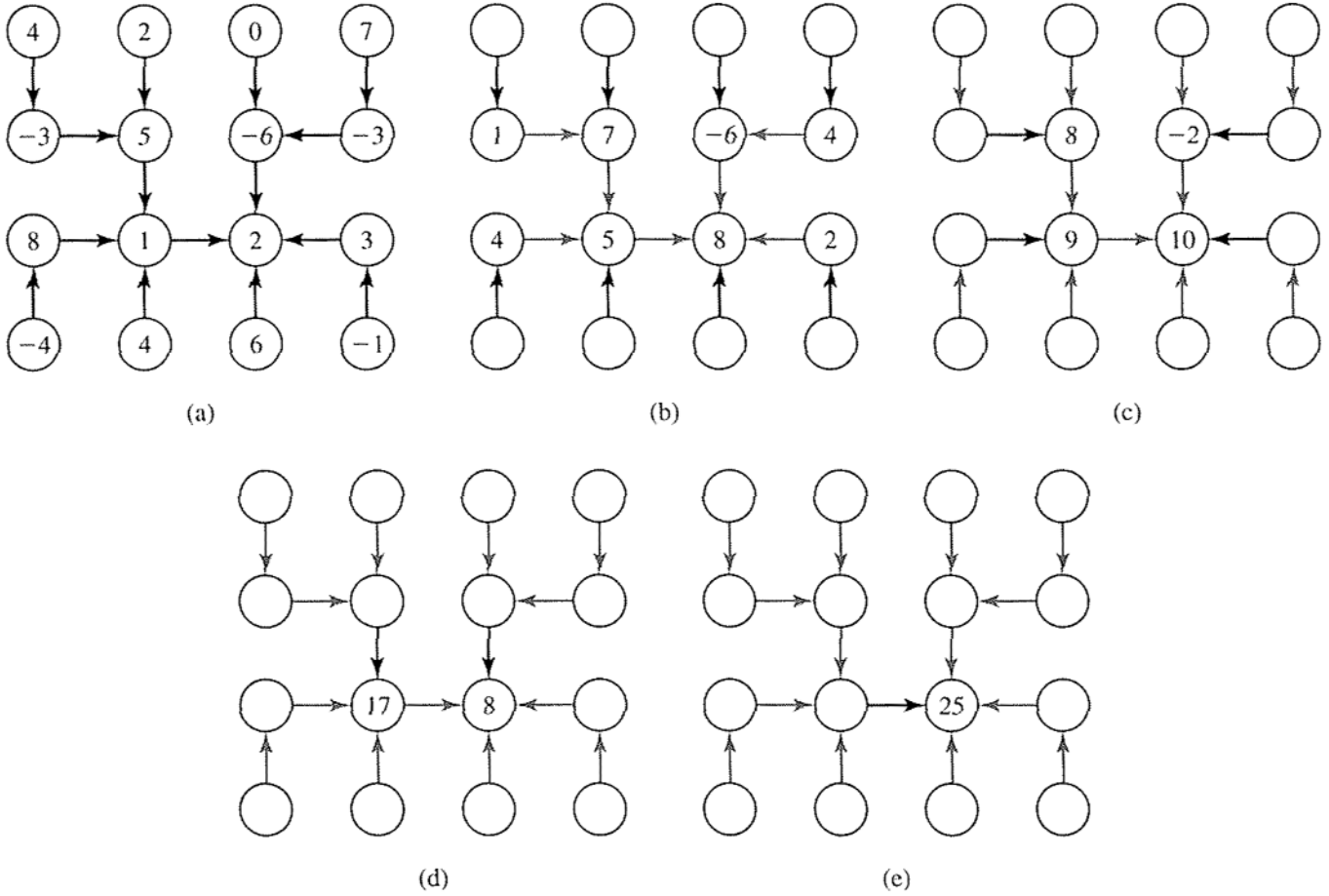


Figure 3.14 Finding the global sum in logarithmic time. (a) A task/channel graph forming a binomial tree. There is one task for each integer value in the list to be added. (b) Half of the tasks send values, and half of the tasks receive values and add. The sending tasks become inactive. (c) A quarter of the tasks send values, and a quarter of the tasks receive values and add. The sending tasks become inactive. (d) The process recurses with two sending tasks and two receiving/adding tasks. (e) In the final step, one task sends and one task receives and adds. The receiving/adding task has the grand total.

Agglomeration and Mapping

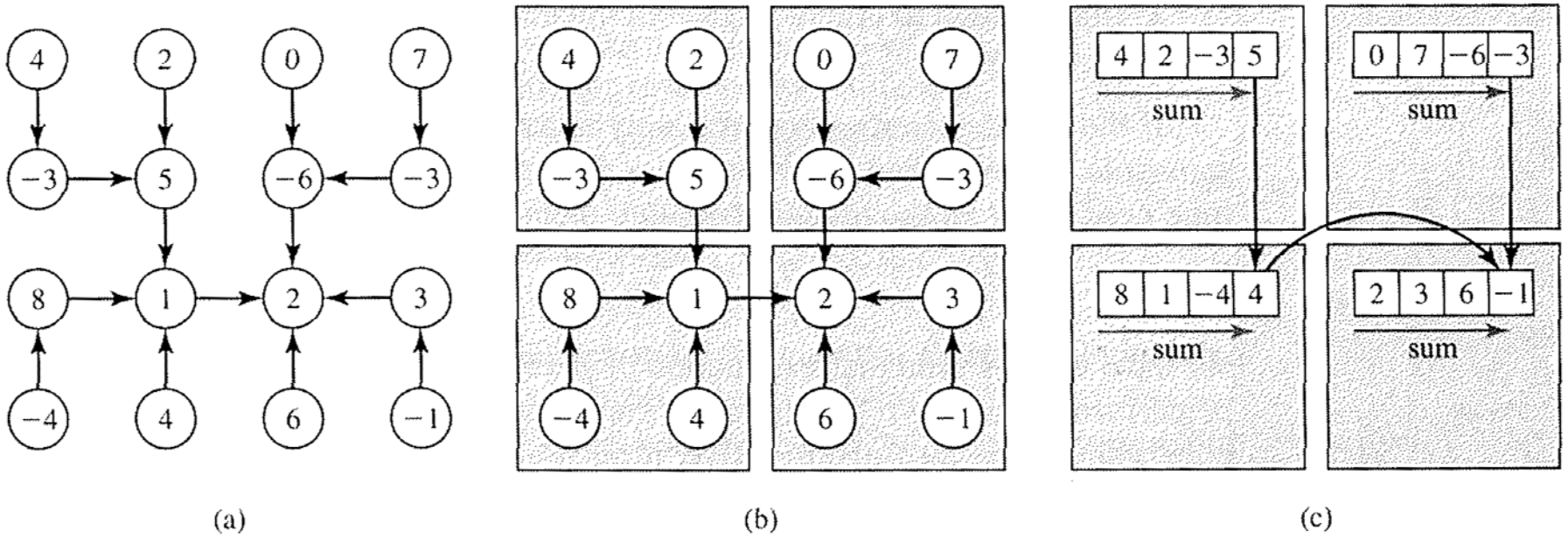
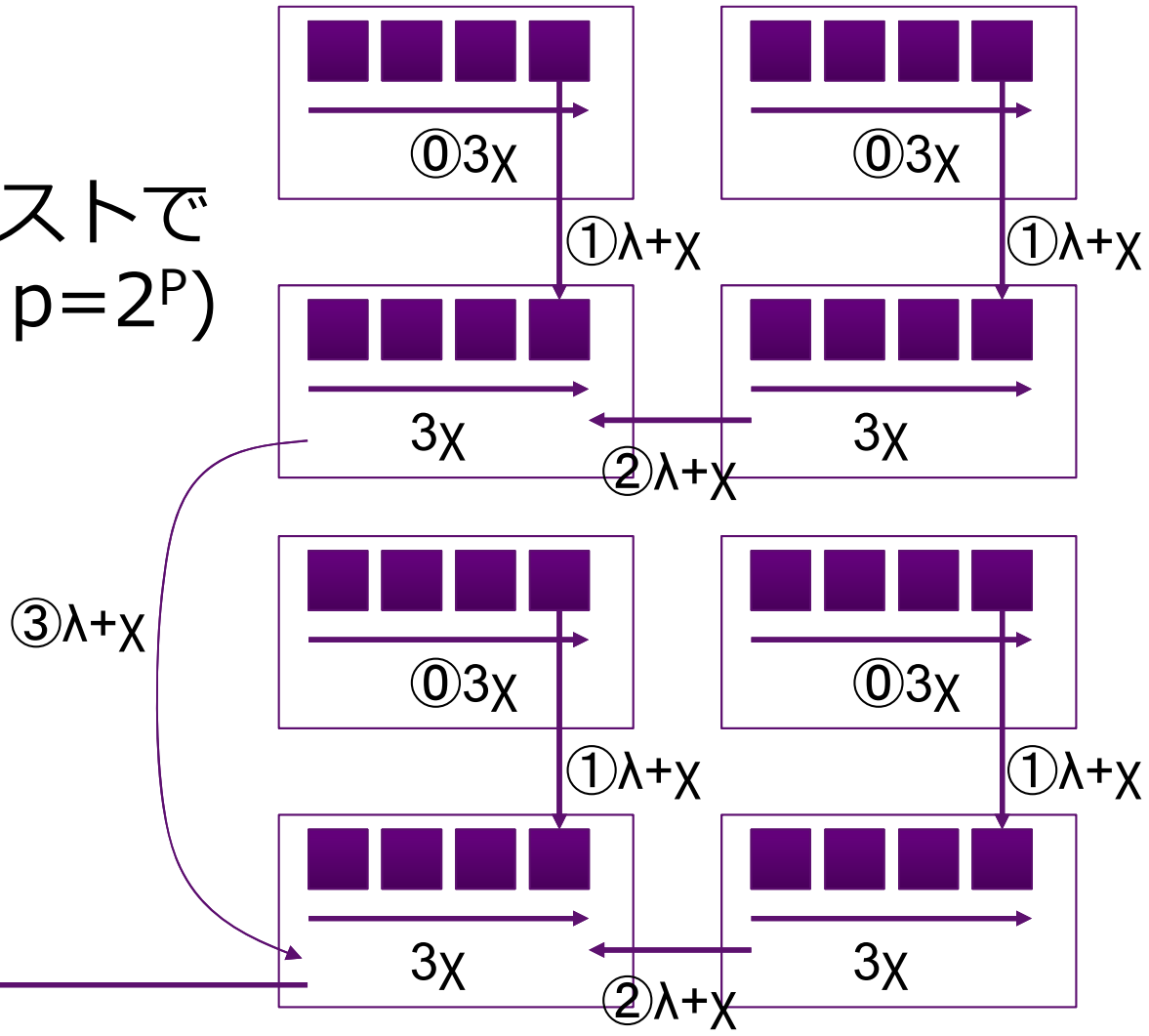


Figure 3.16 Example of agglomeration. (a) The original task/channel graph for the parallel reduction algorithm. (b) Sixteen tasks are mapped to four processors. Each processor has an equal number of tasks, and interprocessor communication is minimized. (c) The four tasks on each processor are agglomerated into a single task. Each task uses the sequential algorithm to find the local subtotal before communicating with the other tasks.

Generalized Model

- n=32, p=8

(今日の理解度テストでは一般形($n=2^N$, $p=2^P$)を扱う)



Total Execution Time
=⓪＋①＋②＋③
＝3λ＋6χ

Answer