HPI Hasso Plattner Institut
Digital Engineering · Universität Potsdam

Parallel Programming and Heterogeneous Computing

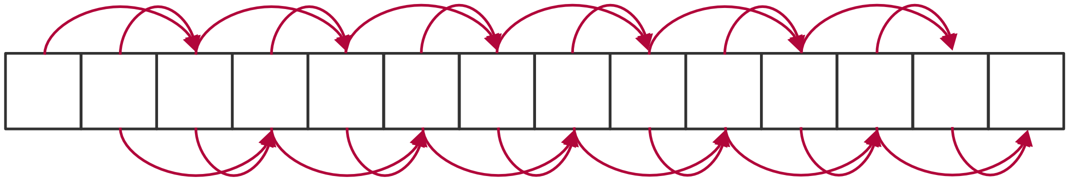
A4 – Workloads & Foster's Methodology

Max Plauth, Sven Köhler, Felix Eberhardt, Lukas Wenzel, and Andreas Polze
Operating Systems and Middleware Group

Example: Can You Easily Parallelize … ?

Computing the *n*-th Fibonacci number:

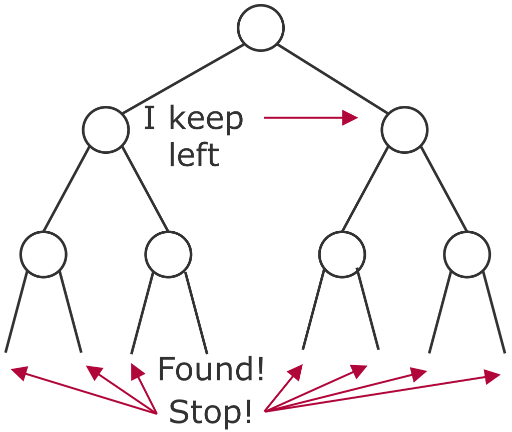
$$F_n = F_{n-1} + F_{n-2}, \text{ with } F_0 = 0, F_1 = 1$$



Cannot be obviously parallelized, due to **data dependency:** the result of one step depends on an earlier step to have produced a result.

Example: Can You Easily Parallelize … ?

Searching an unsorted, discrete problem space for a specific value.



Model space as a tree, parallelize search walk on sub-trees.

Might require communication (“don’t go there”, “stop all”).

Example: Can You Easily Parallelize … ?

Approximating π using a Monte Carlo method?

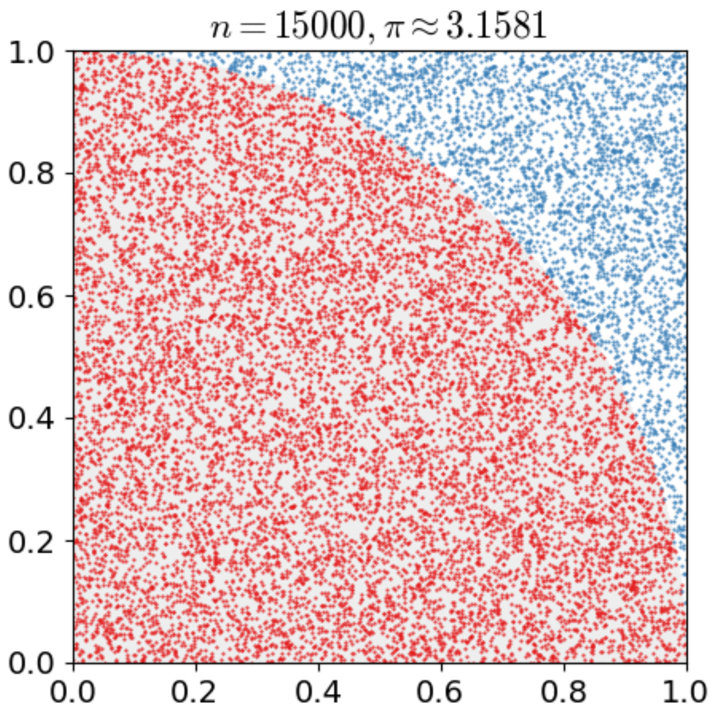
Pick random points $0 \le x, y \le 1$.

Point is in circle if $x^2 + y^2 \le 1$.

P(X): how likely a point ends in X.

$$\pi = 4 * P(\text{circle}) / P(\text{square})$$
$$\approx 4 * \#\text{ptsInCircle} / \#\text{ptsTotal}$$

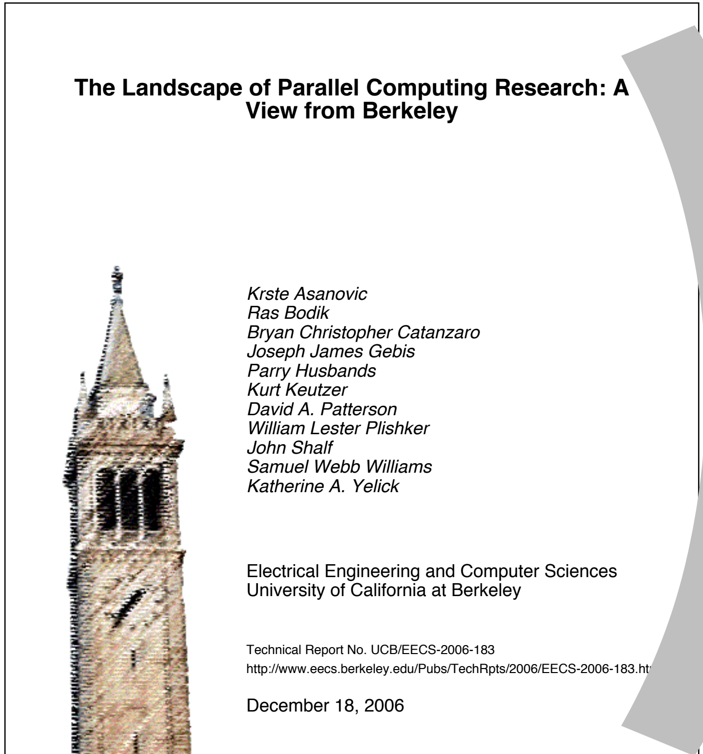
Parallel action for each point completely independend, no commucation required (**embarrassingly parallel**).



Sidenote: Berkeley Dwarfs [Berkeley2006]

Last two slides showed typical examples of different classes of parallel algorithms.

We'll revisit them at the end of this semester, but you can already read up on them.



The Landscape of Parallel Computing Research: A View from Berkeley

Krste Asanovic
Ras Bodik
Bryan Christopher Catanzaro
Joseph James Gebis
Parry Husbands
Kurt Keutzer
David A. Patterson
William Lester Plishker
John Shalf
Samuel Webb Williams
Katherine A. Yelick

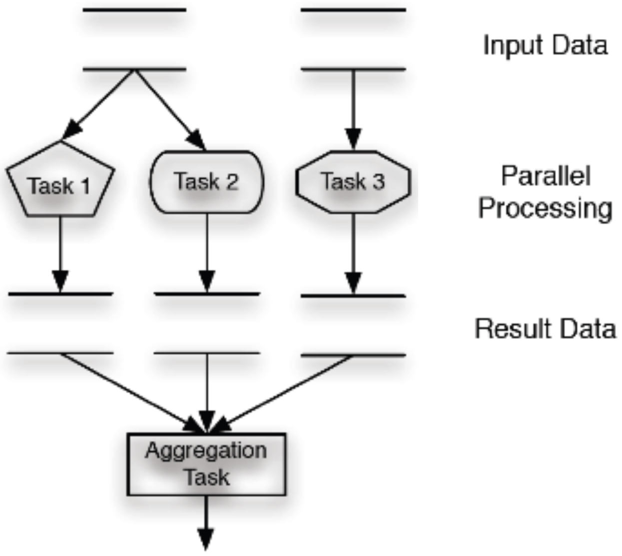
Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2006-183
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.ht

December 18, 2006

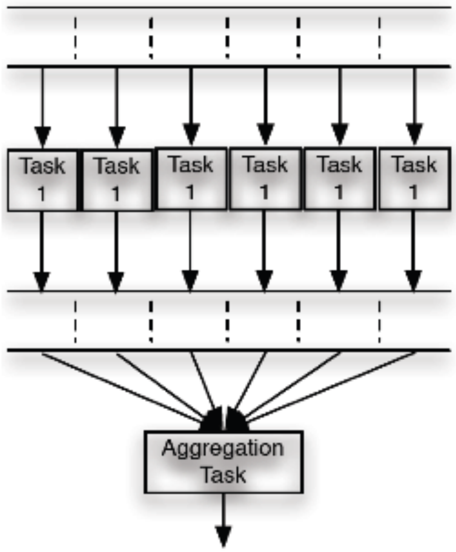
Workloads

"task-level parallelism"



- Different tasks being performed at the same time
- Might originate from the same or different programs

"data-level parallelism"

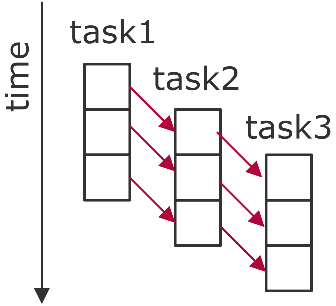


- Parallel execution of the same task on disjoint data sets

Workloads

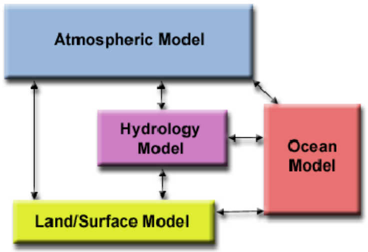
Task / data size can be **coarse-grained** or **fine-grained**.

Size decision depends on algorithm design or configuration of execution unit.



- Sometimes also **"flow parallelism"** added
  - Overlapping work on data stream
  - Examples: Pipelines, assembly line model

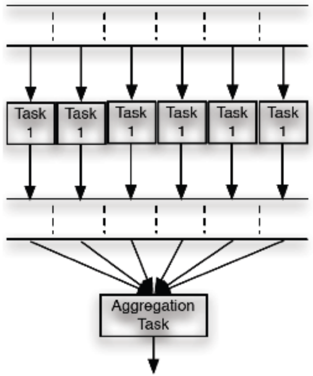
- Sometimes also **"functional parallelism"** added
  - Distinct functional units of your algorithm, exchanging data in a cyclic communication graph



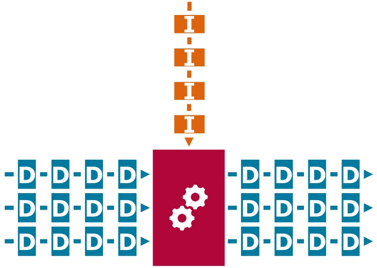
For those four terms no clear distinction in literature.

Execution Environment Mapping

data parallelism



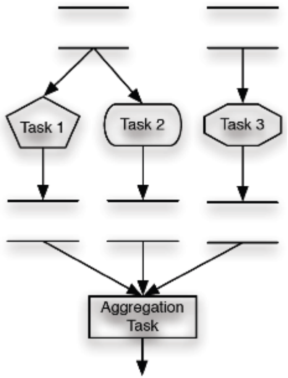
maps well to



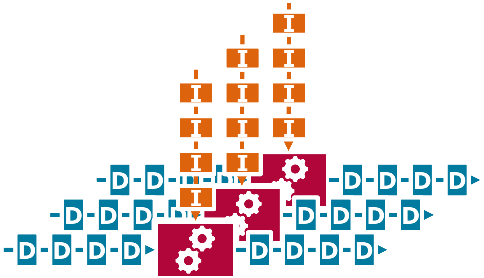
SIMD

Single Instruction stream
Multiple Data streams

task parallelism



maps well to



MIMD

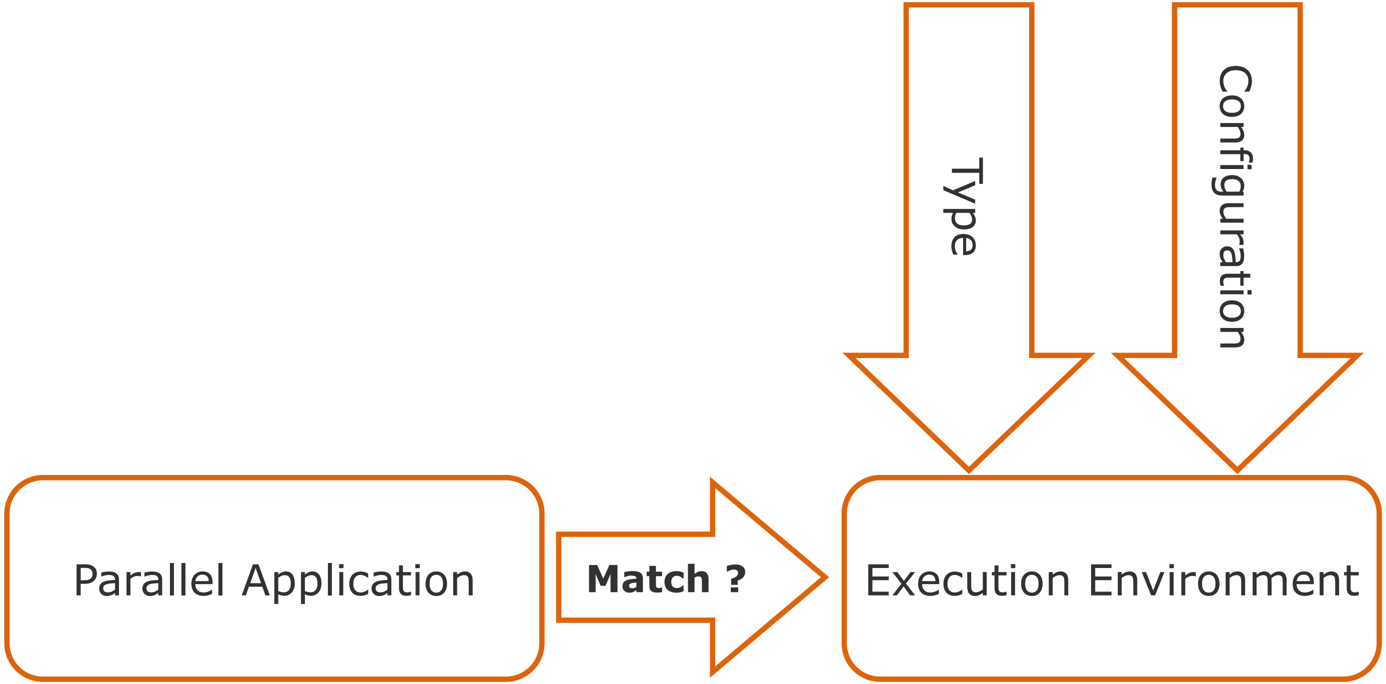
Multiple Instruction streams
Multiple Data streams

Execution Environment Mapping

	Shared Memory (SM)	Shared Nothing/ Distributed Memory (DM)
Data Parallel	**SM-SIMD Systems** *SSE, AltiVec, CUDA*	**DM-SIMD Systems** *Hadoop, systolic arrays*
Task Parallel	**SM-MIMD Systems** *ManyCore/SMP systems*	**DM-MIMD Systems** *Clusters, MPP systems*

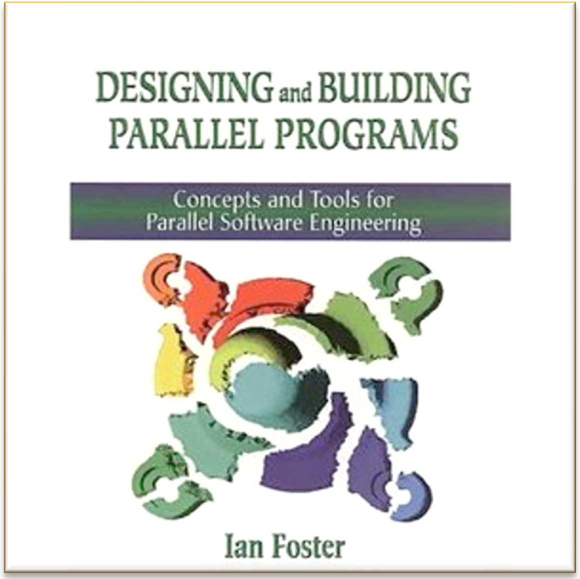
Execution environments are optimized for one kind of workload, event though they can also be used for other ones.

The Parallel Programming Problem



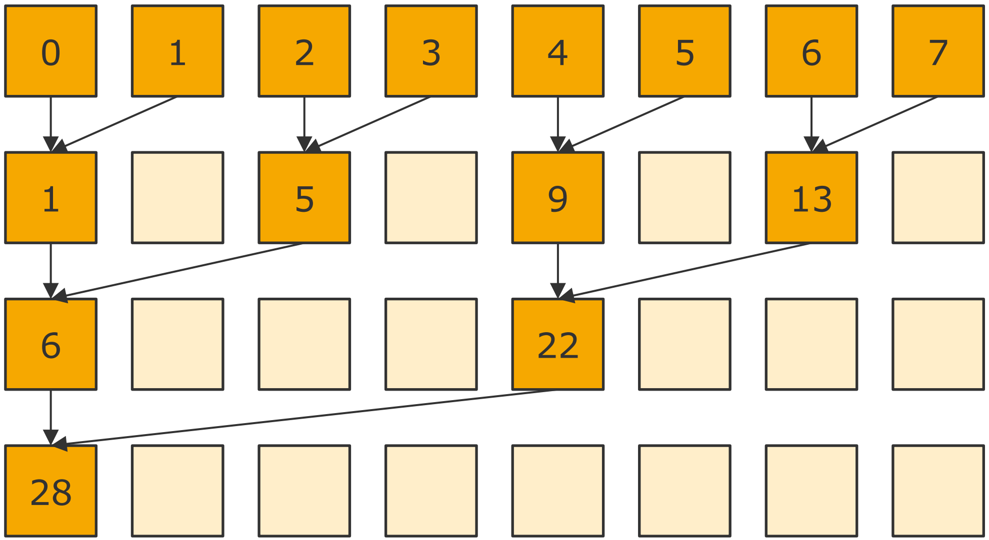
Designing Parallel Algorithms [Foster]

- Map workload problem on an execution environment
  - Concurrency for speedup
  - Data locality for speedup
  - Scalability
- Best parallel solution typically differs massively from the sequential version of an algorithm
- Foster defines four distinct stages of a methodological approach
- We will use this as a guide in the upcoming discussions
- Note: Foster talks about communication, we use the term synchronization instead



Example: Parallel Reduction

- Reduce a set of elements into one, given an operation, e.g. summation: f(a, b) = a + b



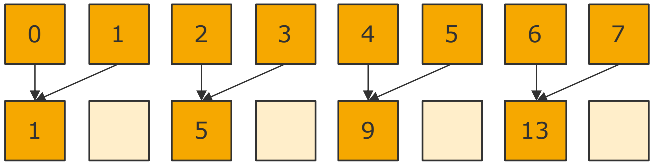
Designing Parallel Algorithms [Foster]

- A) Search for concurrency and scalability
  - **Partitioning**
    Decompose computation and data into the *smallest possible* tasks
  - **Communication**
    Define necessary coordination of task execution
- B) Search for locality and other performance-related issues
  - **Agglomeration**
    Consider performance and implementation costs
  - **Mapping**
    Maximize execution unit utilization, minimize communication
- Might require backtracking or parallel investigation of steps

Partitioning Step [Foster]

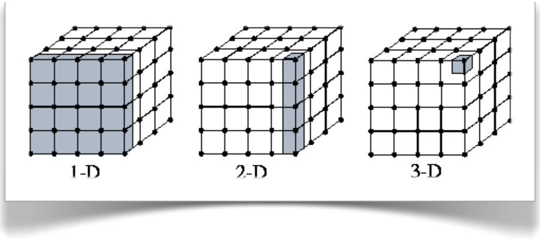
- Expose opportunities for parallel execution – fine-grained decomposition
- Good partition keeps computation and data together
  - **Data partitioning** leads to data parallelism
  - **Computation partitioning** leads task parallelism
  - Complementary approaches, can lead to different algorithms
  - Reveal hidden structures of the algorithm that have potential
  - Investigate complementary views on the problem
- Avoid replication of either computation or data, can be revised later to reduce communication overhead
- Step results in multiple candidate solutions

a + b

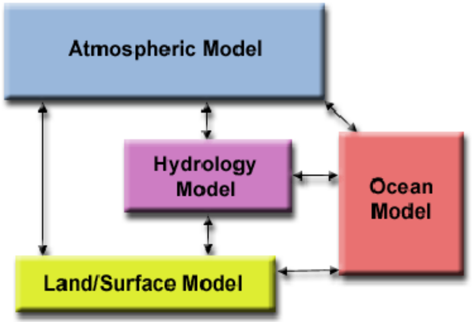


Partitioning - Decomposition Types

- **Domain Decomposition**
  - Define small data fragments
  - Specify computation for them
  - Different phases of computation on the same data are handled separately
  - Rule of thumb: First focus on large or frequently used data structures



- **Functional Decomposition**
  - Split up computation into disjoint tasks, ignore the data accessed for the moment
  - With significant data overlap, domain decomposition is more appropriate



Partitioning - Checklist

- Checklist for resulting partitioning scheme
  - Order of magnitude more tasks than processors?
    -> Keeps flexibility for next steps
  - Avoidance of redundant computation and storage requirements?
    -> Scalability for large problem sizes
  - Tasks of comparable size?
    -> Goal to allocate equal work to processors
  - Does number of tasks scale with the problem size?
    -> Algorithm should be able to solve larger tasks with more processors
- Resolve bad partitioning by estimating performance behavior, and eventually reformulating the problem

Communication Step [Foster]

- Specify links between data consumers and data producers
- Specify kind and number of messages on these links
- Domain decomposition problems might have tricky communication infrastructures, due to data dependencies
- Communication in functional decomposition problems can easily be modeled from the data flow between the tasks
- Categorization of communication patterns
  - Local communication (few neighbors) vs. global communication
  - Structured communication (e.g. tree) vs. unstructured communication
  - Static vs. dynamic communication structure
  - Synchronous vs. asynchronous communication

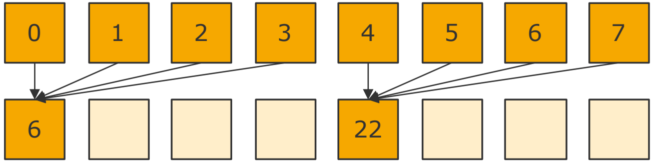
Communication - Hints

- Distribute computation and communication,
  don‘t centralize algorithm
  - Bad example: Central manager for parallel summation
  - Divide-and-conquer helps as mental model to identify concurrency
- Unstructured communication is hard to agglomerate,
  better avoid it
- Checklist for communication design
  - Do all tasks perform the same amount of communication?
    -> Distribute or replicate communication hot spots
  - Does each task perform only local communication?
  - Can communication happen concurrently?
  - Can computation happen concurrently?

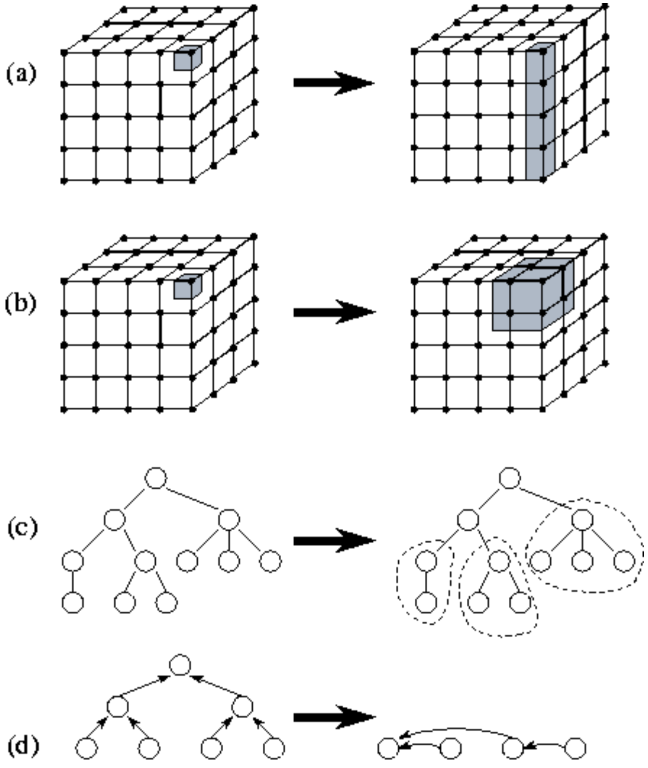
Agglomeration Step [Foster]

- Algorithm so far is correct,
  but not specialized for some execution environment
- Check again partitioning and communication decisions
  - Agglomerate tasks for efficient execution on some machine
  - Replicate data and / or computation for efficiency reasons
- Resulting number of tasks can still be greater than the number of processors
- Three conflicting guiding decisions
  - Reduce communication costs by coarser granularity of computation and communication
  - Preserve flexibility with respect to later mapping decisions
  - Reduce software engineering costs (serial -> parallel version)

`addh4 a,b,c,d`



Agglomeration [Foster]



Agglomeration – Granularity vs. Flexibility

- Reduce communication costs by coarser granularity
  - Sending less data
  - Sending fewer messages (per-message initialization costs)
  - Agglomerate, especially if tasks cannot run concurrently
    - Reduces also task creation costs
  - Replicate computation to avoid communication (helps also with reliability)
- Preserve flexibility
  - Flexible large number of tasks still prerequisite for scalability
- Define granularity as compile-time or run-time parameter

Agglomeration - Checklist

- Communication costs reduced by increasing locality ?
- Does replicated computation outweighs its costs in all cases ?
- Does data replication restrict the range of problem sizes / processor counts ?
- Does the larger tasks still have similar computation / communication costs ?
- Does the larger tasks still act with sufficient concurrency ?
- Does the number of tasks still scale with the problem size ?
- How much can the task count decrease, without disturbing load balancing, scalability, or engineering costs ?
- Is the transition to parallel code worth the engineering costs ?

Mapping Step [Foster]

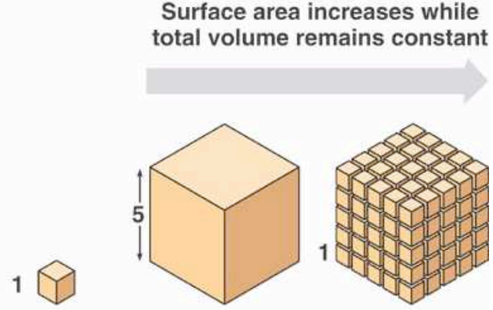
- Only relevant for shared-nothing systems, since shared memory systems typically perform automatic task scheduling
- Minimize execution time by
  - Place concurrent tasks on different nodes
  - Place tasks with heavy communication on the same node
- Conflicting strategies, additionally restricted by resource limits
  - In general, NP-complete bin packing problem
- Set of sophisticated (dynamic) heuristics for load balancing
  - Preference for local algorithms that do not need global scheduling state

Designing Parallel Algorithms [Foster]

- A) Search for concurrency and scalability
  - **Partitioning**
    Decompose computation and data into the *smallest possible* tasks
  - **Communication**
    Define necessary coordination of task execution
- B) Search for locality and other performance-related issues
  - **Agglomeration**
    Consider performance and implementation costs
  - **Mapping**
    Maximize execution unit utilization, minimize communication
- Might require backtracking or parallel investigation of steps

Surface-To-Volume Effect [Foster, Breshears]

Visualize the data to be processed (in parallel) as sliced 3D cube

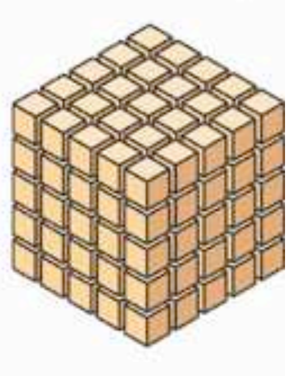


Total surface area (height × width × number of sides × number of boxes)	6	150	750
Total volume (height × width × length × number of boxes)	1	125	125
Surface-to-volume ratio (surface area / volume)	6	1.2	6

[nicerweb.com]

Surface-To-Volume Effect [Foster, Breshears]

- **Synchronization** requirements of a task
  - Proportional to the **surface** of the data slice it operates upon
  - Visualized by the amount of ‚borders' of the slice
- **Computation** work of a task
  - Proportional to the **volume** of the data slice it operates upon
  - Represents the granularity of decomposition
- **Ratio of synchronization and computation**
  - High synchronization, low computation, high ratio → bad
  - Low synchronization, high computation, low ratio → good
  - Ratio decreases for increasing data size per task
- Coarse granularity by agglomerating tasks in all dimensions
  - For given volume, the surface then goes down → good



Literature

[Berkeley2006]

"The Landscape of Parallel Computing research: A View from Berkeley." Asanovic, Krste, et al. (2006) Technical Report No. UCB/EECS-2006-183

[Foster1995]

"Designing and Building Parallel Programs" Foster, Ian (1995) Addison-Wesley

[Breshears2009]

"The Art of Concurrency" Breshears, Clay. O'Reilly Media Inc. 2009

And now for a break and a cup of espresso*.

*or beverage of your choice