Agglomeration

- In this stage, we move from the abstract toward the concrete
  - Tasks are combined so as to produce fewer tasks of larger size
  - Determine whether it is useful to replicate data and/or computation
  - Sometimes it may be best to reduce the number of tasks to exactly the number of processors (i.e., combine the Agglomeration and Mapping stages)
- Important issues to consider now
  - Reducing communication costs by increasing computation / communication granularity
  - Retaining flexibility with respect to scalability and mapping decisions
  - Reducing software engineering costs

Increasing Granularity

- A large number of fine-grained tasks produces flexibility but not necessarily efficiency
  - Communication costs slow down computation and decrease efficiency
- Can communication costs be reduced?
  - Reduce time spent communicating
  - Combine communication into fewer (but larger) messages
  - Combine (agglomerate) tasks that communicate frequently with each other
- Communication and/or execution time can often be decreased by replicating computation

Replicating Computation

- 1-D array to collect & broadcast sum
  - N–1 to collect sum, N–1 to broadcast, total of 2(N–1) steps to get sum to all
- Tree to collect & broadcast sum
  - lg N for sum, lg N for b’cast, 2 lg N steps to get sum to all
  - O(N lg N) computations / communications
- Ring
  - N partial sums in motion simultaneously, N–1 steps to get sum to all
  - (N–1)N computations / communications
  - $(N-1)^2$ redundant comps. / comms.
- Butterfly (see figure 2.14)
  - lg n steps, O(N lg N) operations
  - No broadcast, no redundant operations!

More on Agglomeration

- Preserving flexibility
  - Ability to create varying number of tasks is critical if program is to be scalable
  - Number of processors may change
  - Mapping multiple tasks to one processor allows one task to block during communication, while permitting another task to use that time for communication
  - Still want more tasks than processors to provide flexibility for mapping stage
- Reducing software engineering costs
  - Parallelizing sequential code may best be done by not changing the code any more than necessary
  - Considering parallel program as part of some larger system may force a particular data decomposition, or necessitate a restructuring phase

Agglomeration Checklist

- Reduction in communication costs through increased locality
- Replicated communication: benefits outweigh the costs for a range of problem sizes and processor counts
  - Replicated data: does not compromise scalability by restricting problem sizes and processor counts
- Number of tasks scales with problem size
- Tasks with similar computation and communication costs
  - Sufficient concurrency for current and future target computers
- Smallest number of tasks that does not introduce load imbalances, increase S.E. costs, or reduce scalability

Mapping

- Minimize execution time by either:
  - Place tasks that execute concurrently on different processors
  - Place tasks that communicate frequently on the same processor
- This is an NP-complete problem
  - Domain decomposition with fixed number of equal-sized tasks and structured comm. has straight-forward mapping
  - Domain decomposition with varying work per task or unstructured comm. requires heuristic or probabilistic load balancing
  - Domain decomposition with changing work per task or communication requires dynamic load balancing
  - Functional decomposition yields short-lived tasks that are task-scheduled onto idle processors

Load-Balancing Algorithms

- Used to agglomerate fine-grained tasks from an initial partition into one coarse-grained task per processor
- Recursive bisection
  - Partition into sub-domains of approximately equal size while attempting to minimize communication costs
  - Typically using divide-and-conquer (allows parallel computation)
  - Recursive coordinate bisection
    - Subdivide on longer dimension based on grid coordinates
  - Unbalanced recursive bisection
    - Try different aspect ratios instead of automatically dividing in half
  - Recursive graph bisection
    - Reduce the number of edges crossing sub-domain boundaries

Load-Balancing Algorithms (cont.)

- Local algorithms
  - Avoid the global knowledge required by recursive bisection, use only local info from small number of neighbors
  - Example: compare load to that of neighbors, transfer computation if difference exceeds some threshold
  - Useful but slow to adjust to major changes
- Probabilistic methods
  - Random allocation of tasks to processors
  - Many tasks should equalize load
  - Can require a lot of communication between processors
- Cyclic mappings
  - Each processor is allocated every Pth task
  - May increase communication cost

Task-Scheduling Algorithms

- Used when there are many tasks with weak locality requirements
  - Maintain a task pool, from which tasks are taken for allocation to processors (problems are given to workers to process)
  - Try to minimize communication while also maximize processor utilization
- Manager / worker
  - Central task manager responsible for problem allocation
  - Improve efficiency by prefetching problems and caching problems at workers
- Hierarchical manager / worker
  - Divide workers into disjoint sets, each with a sub-manager
  - Sub-managers communicate periodically to balance the load

Task-Scheduling Algorithms (cont.)

- Decentralized schemes
  - Task pool on each processor, idle workers request problems from other processors (either neighbors, or processors randomly selected)
  - Can also have a central manager that allocates problems in round-robin fashion (bottleneck, but less so than in manager/worker model)
- Termination detection
  - Need a mechanism to determine when search is complete, so idle workers will eventually stop requesting work if there isn't any to perform
  - Easy for a central manager to do, but more difficult in decentralized scheme since there isn't a central record of who is idle, and messages may be in transit

Mapping Checklist

- SPMD design: also consider dynamic task creation and deletion (simpler, problematic performance)
- Dynamic task creation and deletion design: also consider SPMD algorithm (more control, but more complex)
- Centralized manager must not be a bottleneck
- Dynamic load-balancing algorithms: examine different strategies, consider simple probabilistic or cyclic mappings
- Probabilistic or cyclic mappings: need large enough number of tasks to ensure reasonable load balance