Heterogeneous solutions

- ❖ Graphics and Game processors
  - *Graphics Processing Units (GPUs), e.g., NVIDIA and ATI/AMD*
  - *Game processors, e.g., Cell for PS3*
  - *Parallel processor attached to main processor (APU)*
  - *Originally special purpose, getting more general*
  - *Programming model not yet mature*
- ❖ FPGAs – Field Programmable Gate Arrays
  - *Inefficient use of chip area*
  - *More efficient than multicore for some domains*
  - *Programming challenge now includes hardware design, e.g., layout*
- ❖ Tensor processing or other NeuralNet' units for machine learning

Towards parallel computing

- What are suitable performance metrics?
- What is the appropriate parallelism granularity (~ task size)?
- How to benefit from data locality?
- How to map, coordinate, and synchronize tasks?
- How to scheduling and load balance applications?

Parallel programming is much more difficult than sequential programming.

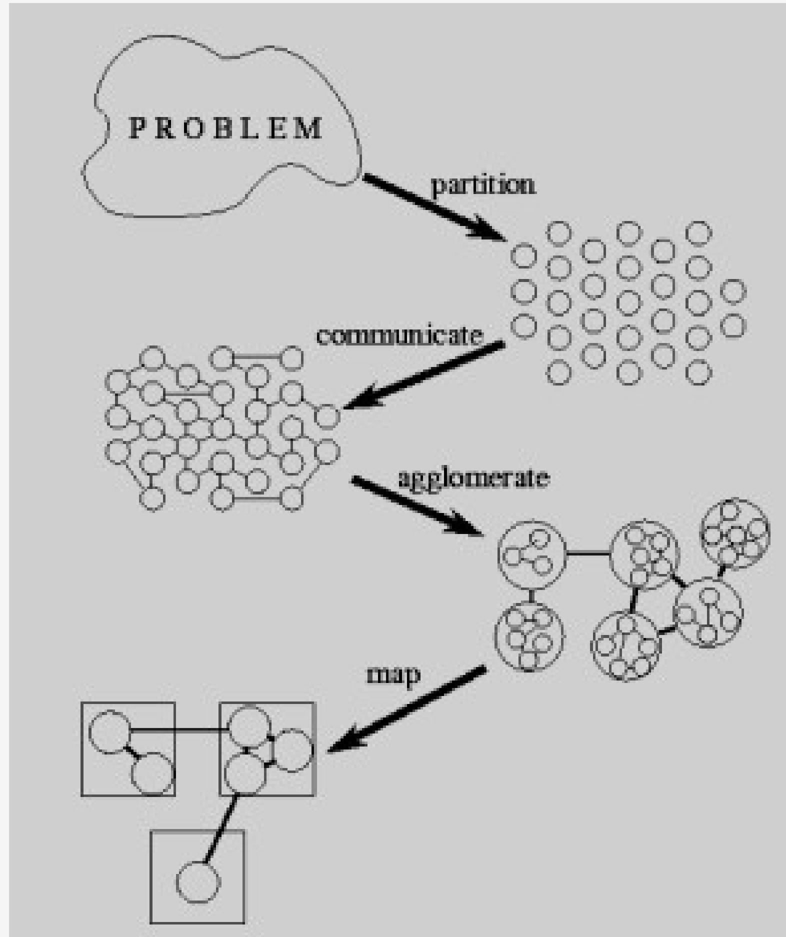
“Invisible” parallelism: exist on every computer

- Bit level parallelism – by **compiler**
  - *e.g., within floating point operations*
- Instruction level parallelism (ILP) – **by compiler/HW**
  - *multiple instructions execute per clock cycle*
- Memory system parallelism – **by compiler/HW**
  - *prefetch, overlap of memory operations with computation*
- Job parallelism – **by OS**
  - *multiple jobs are run in parallel by the OS*

At the parallel application level (parallel tasks and their management), programmers are on their own!

Granularity, mapping

- Partitions = tasks
  - *right size for mapping*
- Communication =>
  - *overhead*
- Data locality =>
  - *agglomerate*
- Mapping =>
  minimize overheads and allow for good load balancing



Beware: agglomerate doesn't always work,
Larger granularity ≠ better data locality

Data locality

- The further the data is from the computation, the slower the computation becomes
- Algorithm should do most work on local data
  - *requires smart data distribution*
- When locality is not possible, use latency hiding techniques
  - *caching and prefetching*
  - *overlapping computation with data transfers*

$$data\ locality\ ratio = \frac{computing\ time}{mem.access \text{ or } communication\ time}$$

Increase data locality by using a data element many times after it is read into cache or communicated from remote (slow) memory.

Challenges

- **Hardware**: processor technology and interconnection network;
- **Software**: Parallel compilers; Grid computing environment (e.g., Condor, Globus, .net, etc.)
- **Methods and Algorithms**: Only computations with large degree of parallelism can efficiently use parallel and distributed computers. (Another important factor is algorithmic efficiency: High performance algorithms are of the same importance as high performance computers!)

Algorithms: Load imbalance

- Workload is imbalanced between processors
  - *less parallelism than processors*
  - *differently sized tasks*
- Can all applications be load-balanced?
  - *adapting to "interesting parts of a domain"*
  - *tree-structured computations*
  - *fundamentally unstructured problems*
- Consequences:
  - *idle resources*
  - *poor hardware utilization*
  - *poor performance*