Writing Parallel Programs

Alexandre David
1.2.05
adavid@cs.aau.dk

Disclaimer

- The only way to learn it is to practice.
  - C/C#/Java syntax similar, same concepts.
  - Once you get it, you can apply that to different languages.
- For those who still complain:
  - AAU educates software engineers & computer scientists, not C# programmers.

Your Practice

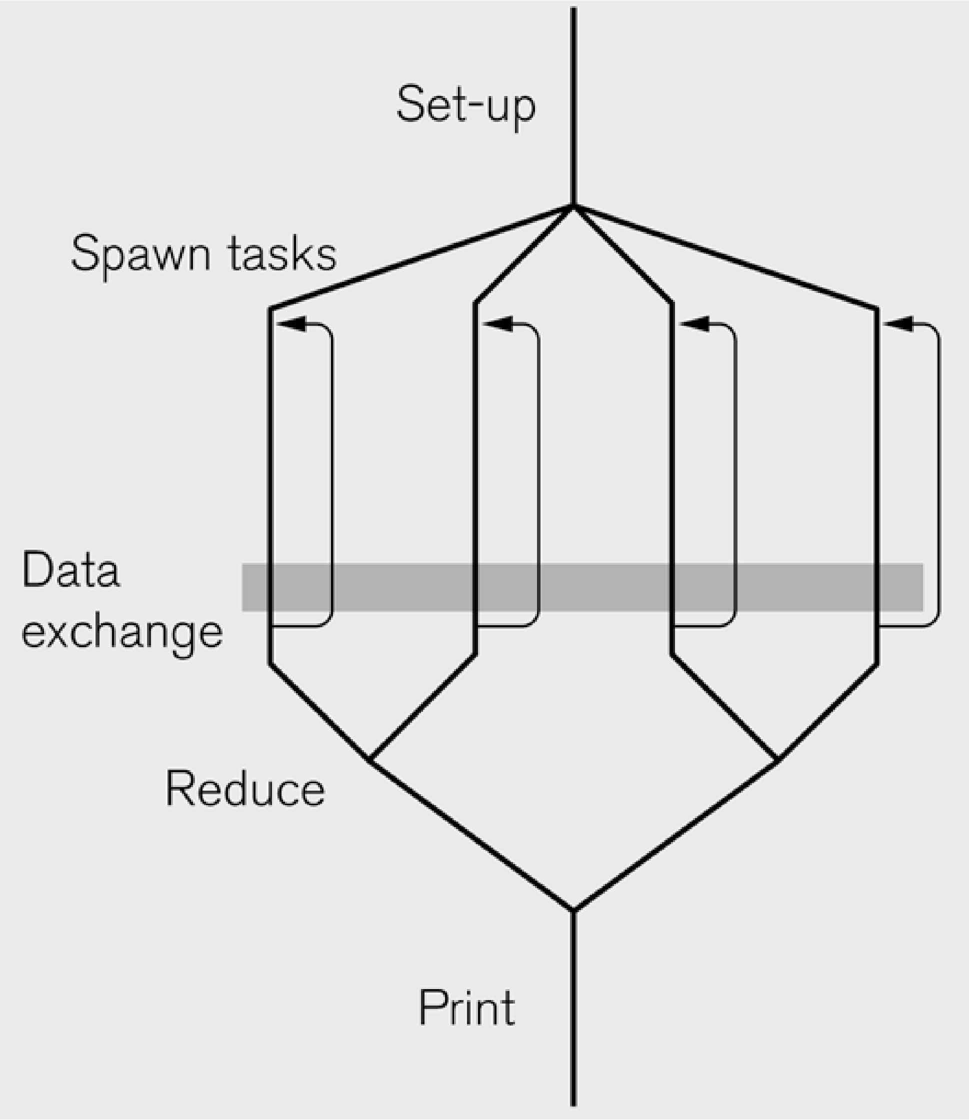
- Assignment 1: Understand influence of caches.
  - Locality is important, avoid false sharing.
- Assignment 2: Basic pthread exercises.
  - Load balancing.
- Assignment 3: More pthread.
  - Practice synchronization.
- Later: MPI, OpenMP.

Recommendations

- Incremental development.
  - Improve working version.
  - Use cvs/subversion.
    - Useful technique: binary search.
- Focus on the parallel structure.
  - Fill in functional parts, don't get lost in details.
  - Possible to debug the parallel parts & the functional parts separately.
- Beware of races.

Example

Possible to develop & test this without specific functional parts.
Parallel interactions are source of complexity & bugs.



Writing Code

- Rule 1: Do not optimize.
  - Write correct code first.
  - Efficiency comes from elegant & simple code.
- Write extra code for testing
  - hooks in functions
  - test generation
  - data inspection
  - periodic checkpoints/log
  - instrument the code to find bottlenecks
  - use assertions

Testing/Debugging

- Get the “right” sized test cases.
  - In general larger tests compared to sequential programs.
- Functional debugging.
- Use modeling tools.

Performance Metrics

- Execution time = time elapsed between
  - beginning and end of execution on a sequential computer.
  - beginning of first processor and end of the last processor on a parallel computer. T_P.

Performance Metrics

- Understand what you are measuring (real/user/sys).
- Compare to the best available sequential algorithm – *don't use p=1*.
  - Speedup $S=T_S/T_P$. *How much faster?* $S \leq p$
  - Efficiency $E=S/p$. *Normalized speedup.* $E \leq 1$

Performance Metrics

- Total parallel overhead.
  - Total time collectively spent by all processing elements = pT_P.
  - Time spent doing useful work (serial time) = T_S.
  - Overhead function: $T_O = pT_P - T_S$.
    General function, contains all kinds of overheads.

Performance Limitation

- Amdahl's Law:
  - Inherent sequential costs will limit speedup.

If a problem of size W has a serial component W_s then
$S \leq W/W_s$ for any p.

Size W corresponds to the serial execution time.
T_P=serial part+(non-serial part)/p
$S=T_S/T_P=W/(W_S+(W-W_S)/p) \leq W/W_S$.

Note: Problem size here = execution time to abstract from particular problem complexities.

Experiments

- Difficult to get consistent results
  - scheduling affect execution time
  - scheduling may affect the results (search problems)
  - average or median (better)
- Don’t conclude too quickly
  - take into account scalability size of the problem + p

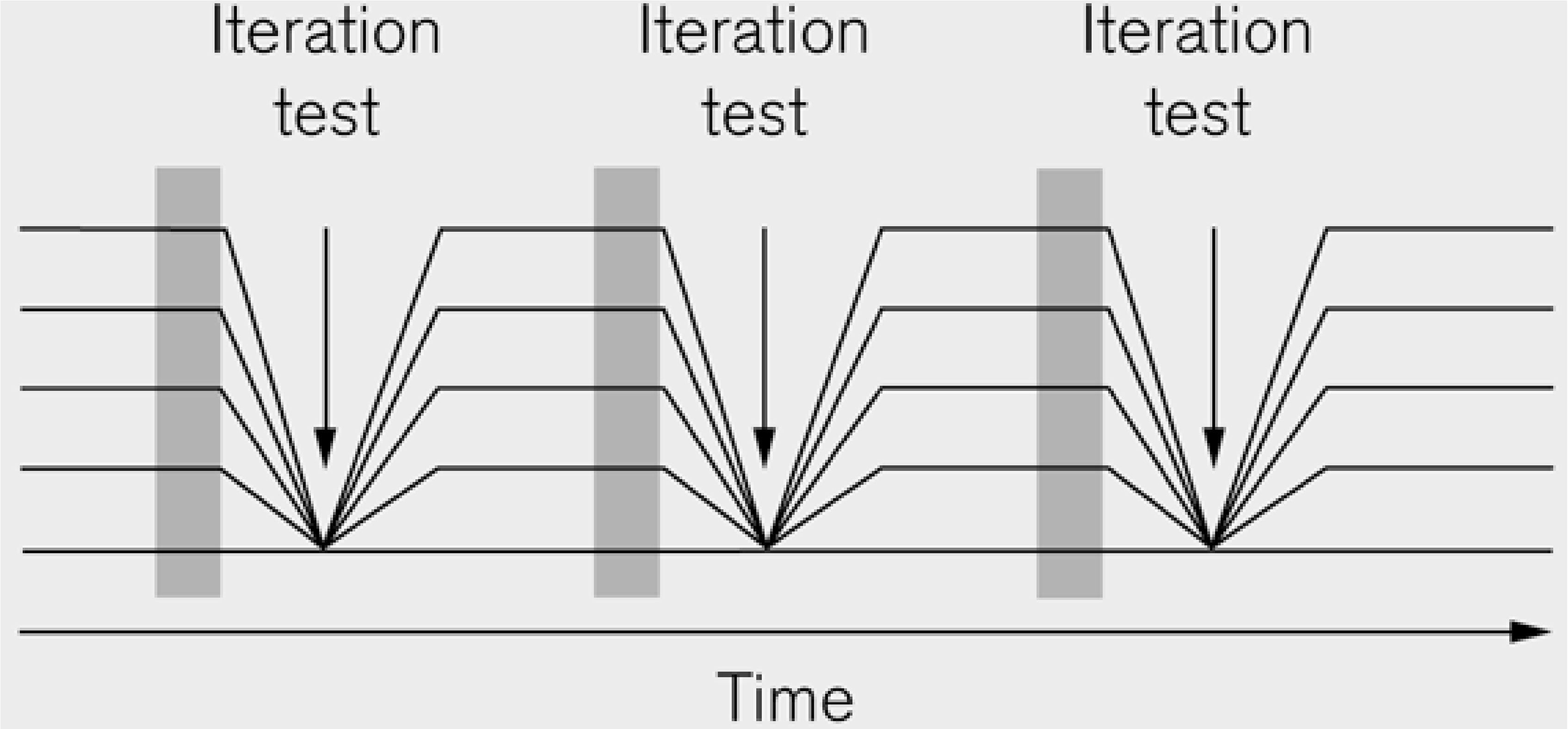
Useful Questions

- What are the individual phases?
  - How to they scale?
  - What are the bottlenecks?
    - How do they synchronize?
  - How much parallelism do we have?
- How much memory is used?
- Are there trade-offs to improve performance? (granularity)

Experimental Methodology

- Hypothesis to find performance bottlenecks
  - load imbalance
  - lock contention
  - communication
  - false sharing
- Emphasize reproducibility.
  - Be aware that instrumentation affects the behavior.

Example



Reductions: Good place to put a barrier to capture the state of the system.