COMP 322: Fundamentals of Parallel Programming

Lecture 4: Parallel Speedup and Amdahl's Law

Instructors: Vivek Sarkar, Mack Joyner
Department of Computer Science, Rice University
{vsarkar, mjoyner}@rice.edu

http://comp322.rice.edu
One Possible Solution to Worksheet 3 (Multiprocessor Scheduling)

There are 4 idle slots in this schedule — can we do better than $T^2 = 15$?

- As before, WORK = 26 and CPL = 11 for this graph
- $T^2 = 15$, for the 2-processor schedule on the right
- We can also see that 
  \[ \text{max} (\text{CPL}, \text{WORK}/2) \leq T^2 < \text{CPL} + \text{WORK}/2 \]
Parallel Speedup

• Define Speedup(P) = $T_1 / T_P$
  — Factor by which the use of P processors speeds up execution time relative to 1 processor, for a fixed input size
  — For ideal executions without overhead, 1 $\leq$ Speedup(P) $\leq$ P
    — This is what you will see with abstract metrics, but these bounds may not hold when we start measuring real execution times with real overheads
  — Linear speedup
    - When Speedup(P) = k*P, for some constant k, 0 < k < 1

• Ideal Parallelism = WORK / CPL = $T_1 / T_\infty$

  = Parallel Speedup on an unbounded (infinite) number of processors
Assume input array size = S is a power of 2, and each add takes 1 unit of time:

- WORK(G) = S-1, and CPL(G) = log2(S)
- Define T(S,P) = parallel execution time for Array Sum with size S on P processors
- Use upper bound $T(S,P) \leq \frac{\text{WORK}(G)}{P} + \text{CPL}(G)$ as a worst-case estimate
  - $T(S,P) = \frac{\text{WORK}(G)}{P} + \text{CPL}(G) = \frac{(S-1)}{P} + \log_2(S)$
  - $\implies \text{Speedup}(S,P) = \frac{T(S,1)}{T(S,P)} = \frac{S}{(S/P + \log_2(S))}$
How many processors should we use?

- Define Efficiency(P) = Speedup(P)/ P = T_1/(P * T_P)
  - Processor efficiency --- figure of merit that indicates how well a parallel program uses available processors
  - For ideal executions without overhead, 1/P <= Efficiency(P) <= 1
  - Efficiency(P) = 1 (100%) is the best we can hope for.

- Half-performance metric
  - S_{1/2} = input size that achieves Efficiency(P) = 0.5 for a given P
  - Figure of merit that indicates how large an input size is needed to obtain efficient parallelism
  - A larger value of S_{1/2} indicates that the problem is harder to parallelize efficiently

- How many processors to use?
  - Common goal: choose number of processors, P for a given input size, S, so that efficiency is at least 0.5 (50%)
ArraySum: Speedup as function of array size, S, and number of processors, P

- Speedup(S,P) = T(S,1)/T(S,P) = S/(S/P + log_2(S))
- Asymptotically, Speedup(S,P) \rightarrow S/log_2 S, as P \rightarrow \infty

Efficiency(P) \leq 0.5, for P \geq 258
   => wasteful to use more than 256 processors for S=2048

Efficiency(P) \leq 0.5, for P \geq 128
   => wasteful to use more than 128 processors for S=1024
Amdahl’s Law [1967]

- If $q \leq 1$ is the fraction of WORK in a parallel program that must be executed sequentially for a given input size $S$, then the best speedup that can be obtained for that program is $\text{Speedup}(S,P) \leq 1/q$.

- Observation follows directly from critical path length lower bound on parallel execution time
  - $\text{CPL} \geq q \times T(S,1)$
  - $T(S,P) \geq q \times T(S,1)$
  - $\text{Speedup}(S,P) = T(S,1)/T(S,P) \leq 1/q$

- This upper bound on speedup simplistically assumes that work in program can be divided into sequential and parallel portions
  - Sequential portion of WORK = $q$
    - also denoted as $f_s$ (fraction of sequential work)
  - Parallel portion of WORK = $1-q$
    - also denoted as $f_p$ (fraction of parallel work)

- Computation graph is more general and takes dependences into account
Illustration of Amdahl’s Law: Best Case Speedup as function of Parallel Portion

Figure source: http://en.wikipedia.org/wiki/Amdahl's law
Functional Parallelism: Adding Return Values to Async Tasks

Example Scenario (PseudoCode)

```java
// Parent task creates child async task
future<Integer> container = async { return computeSum(X, low, mid); };
...
// Later, parent examines the return value
int sum = container.get();
```

Two issues to be addressed:

1) Distinction between `container` and `value` in container (box)
2) Synchronization to avoid race condition in container accesses
Example: Two-way Parallel Array Sum using Future Tasks (PseudoCode)

1. // Parent Task T1 (main program)
2. // Compute sum1 (lower half) & sum2 (upper half) in parallel
3. future<Integer> sum1 = async { // Future Task T2
4.    int sum = 0;
5.    for(int i = 0; i < X.length / 2; i++) sum += X[i];
6.    return sum;
7. };
8. future<Integer> sum2 = async { // Future Task T3
9.    int sum = 0;
10.   for(int i = X.length / 2; i < X.length; i++) sum += X[i];
11.   return sum;
12. };
13. // Task T1 waits for Tasks T2 and T3 to complete
14. int total = sum1.get() + sum2.get();
Future Task Declarations and Uses

• Variable of type future is a reference to a future object
  — Container for return value from future task
  — The reference to the container is also known as a “handle”

• Two operations that can be performed on variable V of type future:
  — Assignment: V can be assigned value of type future
  — Blocking read: V.get() waits until the future task referred to by V has completed, and then propagates the return value
  — Supports waiting on selected tasks, in contrast to finish which waits on all tasks in scope
Announcements & Reminders

- **IMPORTANT:**
  - Watch video & read handout for topic 2.1 for next lecture on Friday, Jan 20th

- HW1 was posted on the course web site ([http://comp322.rice.edu](http://comp322.rice.edu)) on Jan 11th, and is due on Jan 25th

- Quiz for Unit 1 (topics 1.1 - 1.5) is due by Jan 27th on Canvas

- See course web site for all work assignments and due dates

- Use Piazza (public or private posts, as appropriate) for all communications re. COMP 322

- See **Office Hours** link on course web site for latest office hours schedule. Group office hours are now scheduled during 3pm - 4pm on MWF in DH 3092 (default room but alternate room may need to be used on some days — an announcement will be made in the lecture on those days)