Multiprocessors 2014/2015

Simple performance analysis for concurrent programs

Johan Lukkien

Overview

- Performance analysis
  - execution time, speedup, efficiency and scalability
  - communication

Performance metrics

- Different uses
  - problem complexity
    - to what extent does it make sense to search for a parallel solution anyway?
      - e.g., how fast can *N* numbers be added?
    - needs definition of basic computational step [PRAM / CREW]
  - algorithm complexity
    - is this algorithm parallelizable?
      - e.g., Gauss-Jordan does not have the best (sequential) performance but admits a simple, close to optimal parallel solution
    - depends on model of basic computational step [e.g. BSP-lib]
  - analysis program-machine combination
    - empirical, e.g. show problems with mapping; investigate reality level of algorithm complexity
    - basic computational step is implicit, in the used machine and languages

Empirical performance model

- Assume a given, fixed problem
- *T(P,N):* time needed for a size *N* instance on *P* processors
  - each processor spends this time...
  - ... divided in three parts: communication, computation or being idle

- $T(P,N) = c_i(P,N) + b_i(P,N) + y_i(P,N)$, *each processor i*

- When used for *prediction:*
  - effect of topology etc. subsumed in *c* term
  - assumes a homogeneous processor architecture

Penalty

- $T_{seq}(N)$: the time that the fastest (or: a known, fast) algorithm takes for a size N instance
  - on same hardware

- $T_{seq}(N) / T(1,N)$: represents the penalty ρ for choosing this parallel algorithm
  - gives an idea how useful it is: the parallel algorithm on P processors must have an improvement of at least ρ

- Parallel algorithm may do more work
  - e.g. by evaluating $a*b + a*c$ instead of $a*(b+c)$

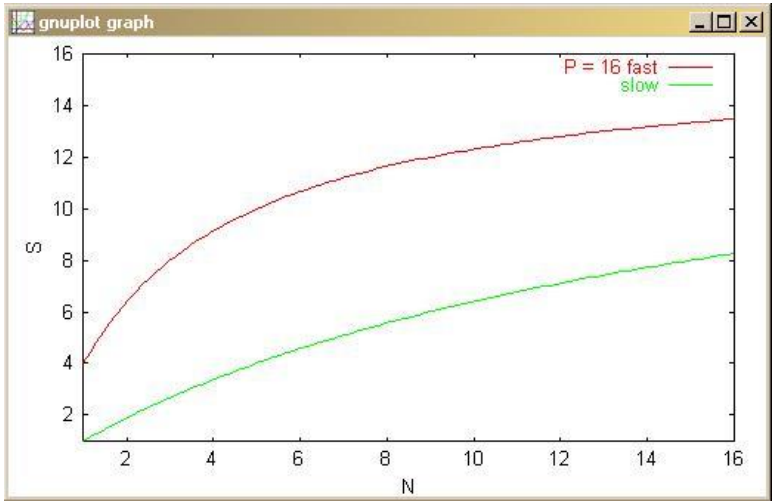
Speedup

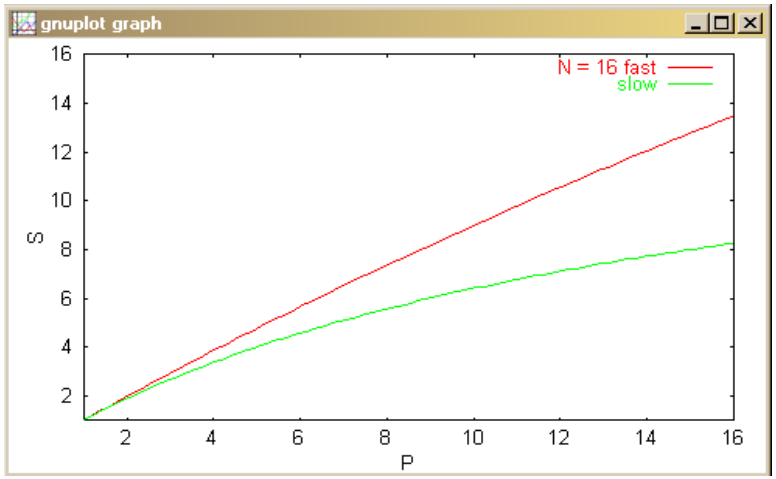
- Speedup represents the improvement in speed
- $S(P,N) = T(1,N) / T(P,N)$
- **Note:**
  - speedup judges the quality of the parallel implementation
  - with respect to the question of *usefulness* of using parallelism, it should be scaled by ρ

Analysis

- Let
  - $B(P,N) = \sum b_i(P,N)$
  - $C(P,N) = \sum c_i(P,N)$
  - $Y(P,N) = \sum y_i(P,N)$
- Then,
  - $P*T(P,N) = B(P,N)+C(P,N)+Y(P,N)$
- Assumption
  - $C(1,N) = Y(1,N) = 0$
- Then,
  - $S(P,N) \quad = \quad P*T(1,N) / P*T(P,N) \quad = \quad P*B(1,N) / (B+C+Y)$ ≤ 1
- Hence,
  - $S(P,N) \leq P$, if $B(1,N) \leq B(P,N)$
  - "super-linear speedup possible only if less work done in parallel case"

Example





- Cubic computation term, quadratic communication term (e.g., matrix operations)
- For large enough problems, concurrency becomes useful
- For small problems only a limited number of machines can be used
- Graphs show fast and slow communication (compared to computation)

Granularity

- *g(N):* represent the duration of the smallest sequential step
  - determined by the algorithm
    - e.g., an inner-product calculation
  - but with a "physical" lowerbound
    - cannot speedup scalar operation

- At least one processor spends this time, hence,

  - $S(P,N) \leq T(1,N) / g(N)$ ["Amdahl's law"]
  - *the number of processors that can be usefully employed to solve a given problem is limited*

    - **notice:** independent of *P*
    - in many cases the actual sequential part is relatively large
    - scalable*: g(N)* increases slower than *T(1,N)*

Efficiency

- Efficiency scales speedup to utilization
- $E(P,N) = S(P,N)/P$

Scaling

- Questions:
  - what relation do I need between processors and problem size to maintain a given efficiency?
    - *E(ie(N), N)* is constant
      - this relation is called the *iso-efficiency*
    - linear and faster: good
    - less than linear: bad
  - I want a given speedup for a given problem
    - how many processors do I need?

Minimizing communication overhead

- Latency hiding
  - do something useful while communicating
    - advance certain communication [look ahead]
    - use concurrent communication hardware
  - use multi-tasking
    - with another concurrent program
      - though this destroys absolute performance
    - increasing the “level of concurrency”
      - several similar processes
      - dedicated communication processes

- Balance (re-)computing, storage and communication
  - e.g caching

- Select an algorithm that
  - admits these techniques
  - has little communication

Overview

- Performance analysis
  - execution time, speedup, efficiency and scalability
  - communication

Simple communication model, no queueing

- Parameters
  - *s:* startup time (includes channel latency)
  - *r*: linkspeed in bytes/sec
  - *n:* message length

- Communication time between two nodes:
  - $t = s + n/r$

- Alternative, use $n_{1/2} = s \cdot r$ as parameter which represents the message length when half the link speed is obtained
  - $t = (n_{1/2} + n)/r$

- Derived parameters
  - effective speed ... $n \cdot r/(s \cdot r+n)$ bytes/sec
  - latency ... *s* seconds

Multiple hops: Store-and-forward

- Transmit message in *m* steps
  - each node receives the message completely
  - and then forwards it

- $t = m \cdot (s + n/r)$ seconds
- Latency: $(m-1)(s+n/r) + s$

Transmission techniques

- Parameters
  - message contains addressing info of size *A*
  - packets have size *P*
  - mapping tables need index size *a*

- Message switched:
  - complete s&f of message
  - $A+n$ bytes are sent

- Packet switched:
  - break message up into fixed sized packets
  - send each packet individually (destination oriented)
  - $P \cdot n/(P-A)$ bytes are sent

Transmission techniques (cnt'd)

- Cut-through
  - the first packet is used to store input/output link relations on all intermediate nodes
  - each packet contains an index in such a table as address information [path oriented]
  - this index is replaced on each hop
  - $P \cdot (A+n)/(P-a)$ bytes are sent

- Wormhole
  - as in cut-through but now the input-output link relation is preserved as a control state
  - links are reserved for the duration of the transmission
  - $A+n$ bytes are sent

Some expressions

- Message switched
  - $t = m \cdot (s+(A+n)/r)$
  - latency: $(m-1) \cdot (s+(A+n)/r) + s$
- Wormhole
  - $(m-1)(s+P/r) + (s+P/r)(A+n)/P$
    - $= (s+P/r)(m-1+(A+n)/P)$
  - latency: $(m-1)(s+P/r) + s$
- Notes:
  - improvement: $m \cdot n$ becomes $m+n$
  - can optimize choice of packet size by computing derivative
    - $dtime/dP = 0$
  - in wormhole, the packet size can be very small
    - what happens if $P=1$?

Finding model parameters

- Start with the simple model
  - $t = s+n/r$
- Estimate parameters
  - vary n
  - least square fit
- Often, two regimes are found
  - small and large messages
- No fit
  - try a more detailed model

Layering

- Computer networks use layered communication
  - units are sent between entities in same layer
    - transmission relies on lower layer
  - entire unit is needed before the next step is made
  - final model is derived through composition

- Example
  - data link communication [frames]
    - use simple model
  - network layer communication [packets]
    - use simple model between pairs
    - results in store & forward at the router level
  - transport layer [segments]