What do you expect from a model?

- A model has to be:
  - extensive:
    - Many parameters – in general, it ends up in a complex system.
    - These parameters reflect the program/machine.
  - Abstract
    - i.e. generic
      - You do not want to change your model each 18 months (see Moore’s law)
      - You want the general trend, not the details.
  - Predictive
    - So it must lead to something you can calculate on.
    - (It does not mean that it must be analytical)

In the parallel world...

- There is no universal model.
- There are many models
  - For each type of machine, and many types of programs.
  - Most theoretical models have been obtained with shared memory models.
    - Much simpler, less parameters.
    - Scalability limited!

Basic ideas for the machine model

- Disconsider the communication (PRAM)
  - Adapted for shared memory machines, multicore chips...
- Consider a machine as being homogeneous, static, perfectly interconnected, with zero latency, and a fixed time for message passing (delay model).
  - Consider a homogeneous, static machine, with a network that has latency and/or a given bandwidth (LogP)
    - Okay for a cluster
  - Consider a dynamic, heterogeneous machine (Grid)...
Parallel program model

• How do you describe a parallel program?
• Task parallelism:
  – The program is made of tasks (sequential units);
  – The tasks are (partially) ordered by dependencies
  – A correct execution if a (possibly parallel) schedule which respects the dependencies.
  – Very close to functional programming.
• The dependencies can be explicit (e.g.: depend on messages) or implicit (e.g.: arguments).
• More complex case: Divide & Conquer.
  – The dependencies order the tasks in a tree-like way.

Programming Model vs. Machine Model

Performance evaluation

• A parallel program is intrinsically non-deterministic
  – The order of execution of the task may change from execution to execution
  – The network (if any) adds its part of random.
• You are interested in runtime.
  – The usual argument “I compiled it, therefore the program is okay” does not serve at all!
  – It is mandatory to use statistical measurements:
    – At least: x runs (x=10,20,30…), and mean, min. and max.
    – Runtime (or speedup, or efficiency) indicated.
    – Better: x runs, mean and standard deviation
    – If the standard dev. is high, run it more – or ask yourself if there is something wrong…
    – Event better: x runs, confidence interval about the mean.

The PRAM model

• A PRAM machine is a set of processors,
  – All are equal and only distinguished by an id.
  – All can access in constant time whatever address of a global, shared memory.
  – The processors execute their instructions synchronously.
  – You can use as many processor as you want.
• Metrics: executing a parallel program with entry of size \( n \), on a PRAM machine, is characterized by two quantities:
  – The parallel runtime \( T_{\text{par}}(n) \)
  – The number of processors required to this execution \( P(n) \)
• The “quality” of the PRAM execution is also measured by \( W_{\text{par}}(n) \) (Work), the total number of instructions.
  – \( W_{\text{par}}(n) = T_{\text{par}}(n) \times P(n) \)

Considerations: time, space and work

• \( T_{\text{par}}(n) \) is the runtime.
  – Proportional to the runtime of a single instruction.
  – What is important is the order of magnitude:
    – \( O(n), O(\log n), O(\log \log n), O(n) \)
• \( P(n) \) is the number of processors.
  – In the PRAM model, 1 processor = 1 process.
  – \( P(n) \) can also be considered as a measure of the (memory) space that is required.
• \( C(n) = T_{\text{par}}(n) \times P(n) \) is the (parallel) cost.
  – Look at it as a rectangular area.
• \( W_{\text{par}}(n) \) is the Work
  – A sub-area of the rectangle.
  – As close as possible as the sequential program.
Optimal PRAM algorithm

1.  $T_{par}(n)$ as small as possible
   - Maybe you will have to use many processors!
   - What is small?
     - $T_{par}(n) = \Theta(\log n)$

2.  $P(n)$ not too big.
   - Polynomial in $n$.

3.  Do not perform (many) more instructions in parallel than in sequential.
   - i.e. $C(n) = \Theta(W_p(n)) = \Theta(W_p(n)) = \Theta(T_p(n))$

Great, but what does it mean?

- Take an optimal PRAM algorithm
  - Very parallel, runtime much smaller than seq. and almost as few instr. as in the sequential case.
  - Formally, $T_{par}(n) = \Theta(\log n)$ and $W_p(n) = \Theta(T_p(n))$

- Therefore, you can always run it on a fixed, small number of processors $p$, with runtime:
  $$T_p(n) \leq \frac{T_s(n)}{p} + \log(n)$$

- So, when $T_s(n) \gg \log(n)$ (which is always the case...), you end up with an almost linear speedup. Nice!

1st PRAM algorithm: sum of $n$ elements

- Input: an array of $n = 2^b$ elements and an associative, commutative operator $+$.
- Output: the “sum” of the $n$ elements.
- $T_s(n) = n-1$ /* I do hope this is obvious for everyone... */
- Parallel algorithm: binary tree.

The optimal parallel sum algorithm

- The problem comes from a very fine-grained algorithm.
  - The basic operation is a single $+$ operation.
- Other version of the (same) problem: we use a little bit more processors than what we want.
  - If we could use $P(n) = n/\log(n)$, with $T_p(n) = \Theta(\log n)$, then the algorithm would be optimal.
- Solution: Increase the granularity, or (same thing) use Brent’s principle.
  - Take $p = n/\log(n)$ processors, each one running more than one of the basic $+$ instructions of previous algorithm.
  - Each processor will run $\log(n)$ instructions.
- This idea can also be seen as a sequential “degeneration” of the parallel algorithm.
  - To be efficient in parallel, run in sequential!
  - A similar technique is used in sequential algorithmics (see quicksort)

PRAM complexity of the parallel sum

1.  $T_{par}(n) = \Theta(\log n)$
   - Good.

2.  $P(n) = n/2$
   - That is a "small" polynomial in $n$. Good.

3.  $W(n) = n/2 + n/4 + n/8 + ... + n/2^b = n-1$
   - $= T_p(n)$, great.

4.  So what’s wrong ?
   - $C(n) = P(n) \cdot T_{par}(n) = \Theta(n \log n) \gg \Theta(T_p(n))$
   - In plain English: the $P(n)$ processors are under-used. The algorithm is inefficient.
The optimal parallel sum algorithm

Parallel Prefix

- Input: an array of \( n \) elements and an associative, commutative operator \( + \).
- Output: the \( n \) “partial sums” of the first elements, \( i = 1 \ldots n \).

\[ T(n) = n \]

- I do hope this is obvious for everyone.

\[ \text{result}[1] = a[1] \quad \text{the 1st element of the input array}\]

\[ \text{for} \ (i = 2; i <= n; i++) \]

\[ \text{result}[i] = \text{result}[i-1] + a[i] \]

- This seems highly sequential!
- Let us revisit the computation, using Divide & Conquer
  - This is an IMPORTANT (although simple) concept.

Prefix – the D&C version

\[ C(n) = P(n) \times T \]

Parallelism of the D&C parallel prefix

1. \( T_{\text{par}}(n) = T_{\text{par}}(n/2) + 1 = \ldots = \Theta(\log n) \)
   - Good.

2. \( P(n) = \text{Max} (2P(n/2) ; n/2) = \ldots = n \)
   - That is a “small” polynomial in \( n \). Good.

3. \( C(n) = P(n) \times T_{\text{par}}(n) = \Theta(n \log n) \gg \Theta(T(n)) \)
   - In plain English: the \( P(n) \) processors are under-used. The algorithm is inefficient.

4. “Apply Brent” – or increase the granularity – and you will get an optimal version.
   - \( T_{\text{par}}(n) = \Theta(\log n), P(n) = n/\log(n) \).

Sum of two n-bits numbers.

- Input: 2 binary numbers \( a \) and \( b \) of \( n = 2^k \) bits.
- Output: the \( n+1 \) bits number equal to \( a+b \).
- \( T(n) = n \), with the algorithm that is learned at elementary school (sum the digits column by column, from right to left, with the carry).

\[ \text{Carry} \ c_i = 0 \text{ or } 1, \text{ depending on } c_i \]

Simple and nice, but highly sequential again!
- You have to propagate the carry from right to left, and can not compute the i-th bit without the carry.
Conclusion about PRAM

- A simple, but powerful model
  - Quantifies the runtime and the processor number.
  - Evaluates the parallel number of operations.
  - Provides complexity classes (NC).
- An unrealistic model?
  - Use as many processors as you want
  - This is an approximation – “Brent resolves the problem”.
  - Homogeneous architecture
  - Ok...
  - Uniform time for all the memory accesses
  - This is a real problem! What if there is some network activity in some place?
- Some say that the new area of shamed-memory systems (multicore) give a new force to PRAM.