

# ACM/CS 114

## Parallel algorithms for scientific applications

Michael A. G. Aïvázis

California Institute of Technology

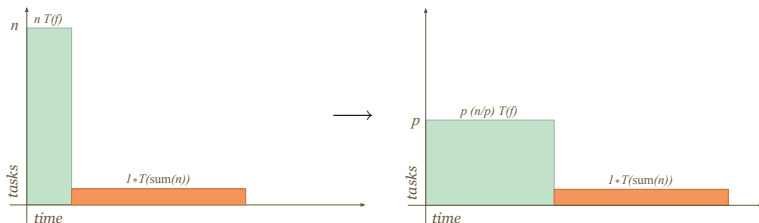
Winter 2010

# Time, parallelism and computational work

- recall our embarrassingly parallel reduction:
  - given a function  $f$  and a sequence of numbers  $S$  of length  $N$ , evaluate

$$s = \sum_{i=0}^{N-1} f(S_i)$$

- initial parallelism profile for a simple mapping, assuming that
  - the computation of  $f(S)$  is the parallel task
  - the summation is sequential



- shaded area is  $w$ , the *computational work*

# Metrics: speedup and efficiency

- ▶ let
  - ▶  $T_1$  be the sequential execution time on one processor
  - ▶  $T_p$  be the parallel execution time on  $p$  processors

- ▶ define

- ▶ *speedup*:

$$\sigma := T_1/T_p$$

- ▶ *efficiency*:

$$\eta := T_1/(pT_p)$$

- ▶ related through  $\eta = \sigma/p$  and  $\sigma = \eta p$

- ▶ pseudo-theorems:  $\sigma \leq p$  and  $\eta \leq 1$

- ▶ but *speedup anomalies* can occur if resources increase with  $p$  causing an increase in the effective computation rate
  - ▶ example: for large enough  $p$ , your problem may fit entirely in the L2 cache
  - ▶ *sweet spots* like that abound; the craftsman knows how to
    - ▶ implement the solution in a portable manner
    - ▶ expose enough controls to be able to tune the implementation to a given architecture

## The bad news: Amdahl's law

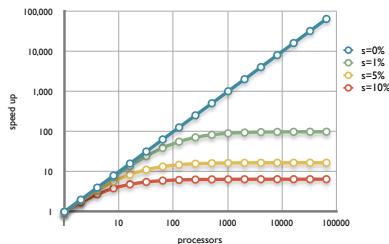
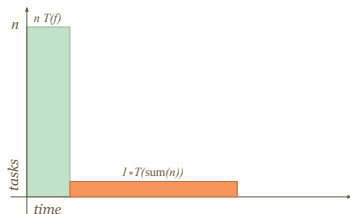
- ▶ consider a solution that consists of two parts
  - ▶ a serial fraction  $s$  with  $0 \leq s \leq 1$
  - ▶ a  $p$ -fold parallel fraction  $1 - s$
- ▶ for a fixed problem size, Amdahl's law relates  $T_p$ ,  $\sigma$  and  $\eta$  to  $T_1$  and  $s$

$$T_p = sT_1 + (1-s)T_1/p$$

$$\sigma = \frac{p}{sp + (1 - s)}$$

$$\eta = \frac{1}{sp + (1-s)}$$

- with corollaries  $\sigma_\infty = \frac{1}{s}$  and  $\eta_\infty = 0$



# Beating Amdahl's law

- ▶ Amdahl's law holds if either
  - ▶ the problem size is fixed
  - ▶ the serial fraction  $s$  is not a function of  $p$
- ▶ *weak scaling*: let the problem size grow with  $p$ 
  - ▶ larger computers are used to solve larger problems
  - ▶ the effective serial fraction *decreases* with problem size
  - ▶ the right scaling metric would be constant, or properly bounded, as  $p \rightarrow \infty$
- ▶ *isoefficiency*
  - ▶ how rapidly must problem size grow so that  $\eta$  is constant as  $p$  increases?
  - ▶ since  $\eta = T_1/(pT_p)$ , constant efficiency implies

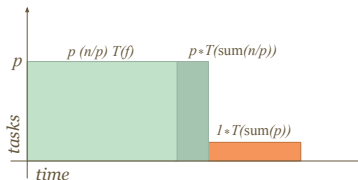
$$T_1 = c(pT_p)$$

for some constant  $c$

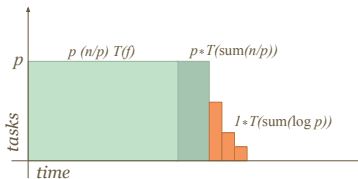
- ▶  $T_1$  measures the sequential work, so the above relation determines your implementation's *isoefficiency function*

# Algorithmic improvements

- ▶ getting smarter is the best way to improve  $\sigma$  and  $\eta$ 
  - ▶ reduce the sequential fraction  $s$
  - ▶ what are the effects on communication and locality?
- ▶ parallelize the partial sums

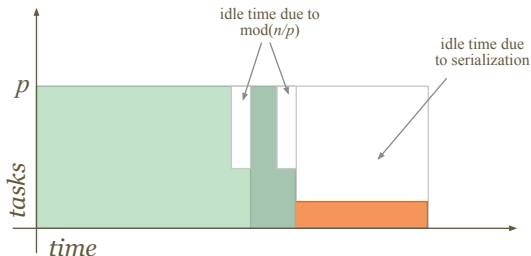


- ▶ parallelize the final sum using a *reduction tree*



# Load balance

- ▶ non-optimal task distributions show up as *load imbalance*

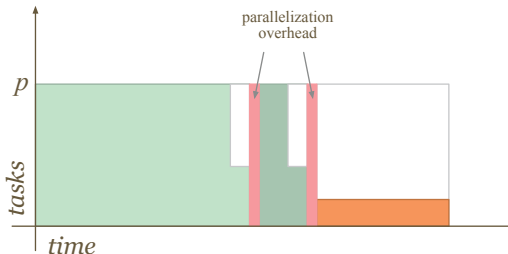


- ▶ excessive coarsening tends to increase load imbalance
- ▶ so can inappropriate mapping
- ▶ synchronization also causes load imbalance (see later slide)
- ▶ new upper bound for the speedup

$$\sigma \leq \frac{w_1}{\max_p(w_p + \text{idle})}$$

# Parallelization overhead

- ▶ there is always some extra work that is not present in the sequential implementation



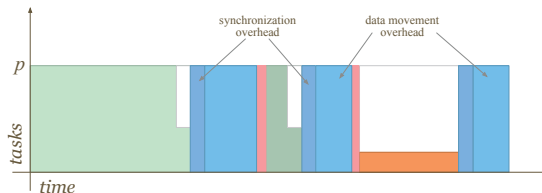
- ▶ orchestration, management, bookkeeping

$$\sigma \leq \frac{w_1}{\max_p(w_p + \text{idle} + \text{overhead})}$$



# Communication and synchronization costs

- communication is required for data movement and synchronization



- the cost is modeled by

$$T_c = \lambda + \beta L$$

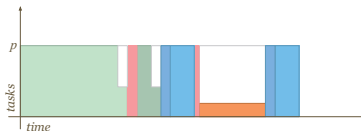
where the *latency*  $\lambda$  measures the communication startup cost,  $\beta$  is the bandwidth of the interconnect and  $L$  is the message length in *words*

- the speedup is now bounded by

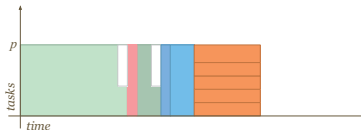
$$\sigma \leq \frac{w_1}{\max_p(w_p + \text{idle} + \text{overhead} + \text{comm})}$$

# Reducing communication costs

- ▶ multiple strategies
- ▶ coördinating placement of work and the associated data to minimize inter-process dependencies



- ▶ trading memory for efficiency by replicating data
- ▶ trading cpu for efficiency by doing redundant work



- ▶ improving communication efficiency by tuning the cost factors
  - ▶ communication frequency, message size, contention, architecture specific optimizations

# Optimizing speedup and efficiency

- ▶ the goal is to minimize the denominator

$$\sigma \leq \frac{w_1}{\max_p(w_p + \text{idle} + \text{overhead} + \text{comm})}$$

- ▶ but its parts are in tension: minimizing one happens at the expense of another
- ▶ fine grain decomposition and intelligent mapping tend to minimize load imbalance at the cost of increased communication
  - ▶ coarser grains imply larger message size and fewer synchronization events
  - ▶ for many problems communication costs decrease as surface to volume
- ▶ naïve static partitioning reduces redundant work but cause load imbalance

# The good news

- ▶ the basic work unit of a parallel algorithm may be more efficient (and better performing) than the sequential equivalent
  - ▶ only a small fraction of typical problems fits in L2 cache
  - ▶ single node performance *requires* partitioning
  - ▶ just like the parallel implementation
  - ▶ don't be surprised by the poor quality of your sequential version after you see your parallel implementation
- ▶ communication can be interleaved with computation
  - ▶ better algorithms on today's complicated memory hierarchies
- ▶ parallel algorithms may lead to better sequential ones
  - ▶ e.g. parallel search may explore configuration space more effectively