# Skeletal Programming
# Present & Future

**Murray Cole**

**University of Edinburgh**

# Why are we here this week?

Conceptually, there must be more to parallel programming than the choice of an appropriate collection of “scalar” interaction primitives.

Because abstraction is good and it should have something to say about parallel program/algorithm design.

Patterns of parallel control and interaction abstract to second order functions.

# Key Issues

- expressivity
- cost calculus
- role of formal methods
- doing it!

# Expressivity

- Choosing the right types and operations
  - Saying enough but no more
  - Generic (“ordinary languages”) or application oriented?
  - Base level paradigm?
- Some consensus?
  - map, fold, scan
  - d&c, farm, pipe, all-pairs
  - operational v. denotational divergence

# Divide & Conquer

- `d&c` is a 'good idea'

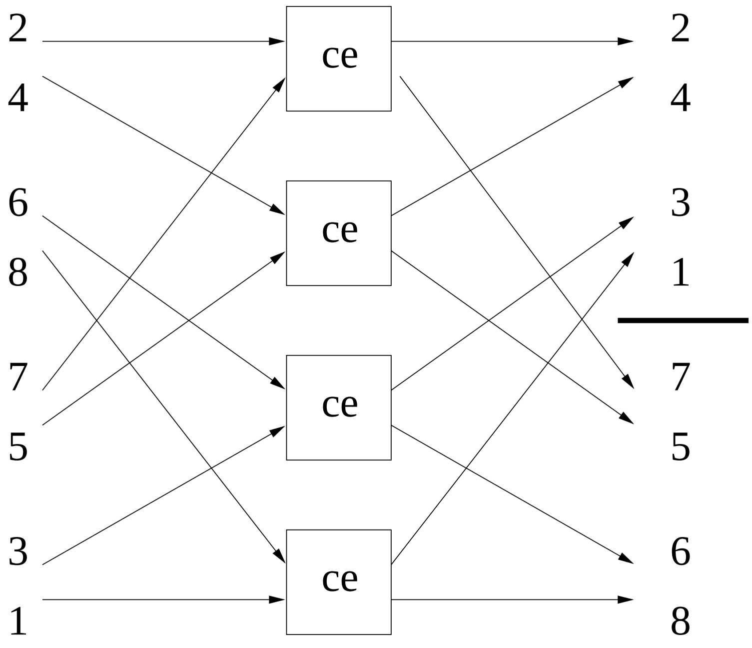
```
d&c t b d c x =
  if t x then b x
  else c (map (d&c t b d c) (d x))
```

- what if it is too flexible?

- bitonic mergesort as nested `d&c` (Mou)

- but the outer 'divide' and inner 'conquer' are conceptually redundant

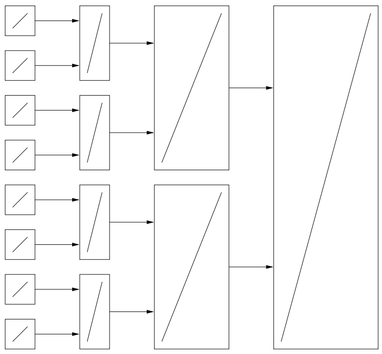
# Bitonic merge step




- all losers $\leq$ all winners

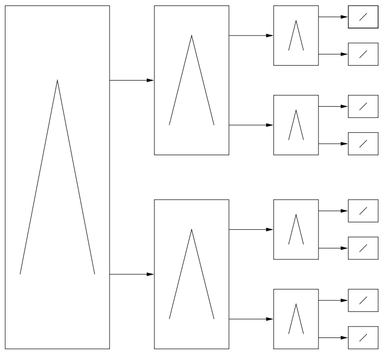
- both sequences independently bitonic

# Outer level - merge tree



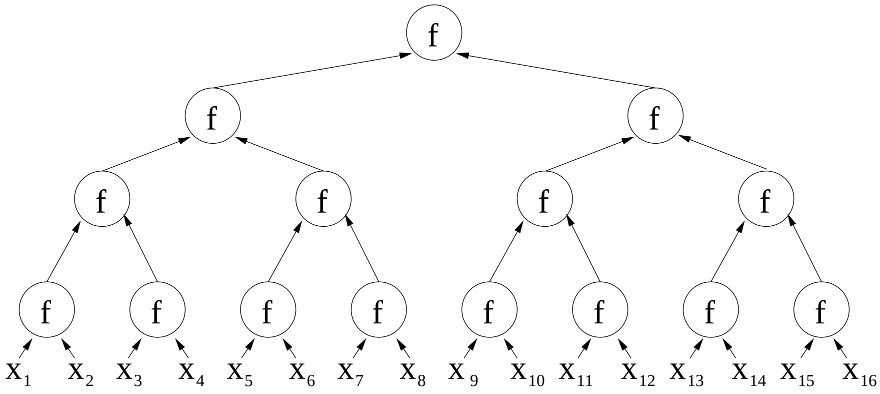
a conquer with no divide

# Inner level - bitonic merge tree

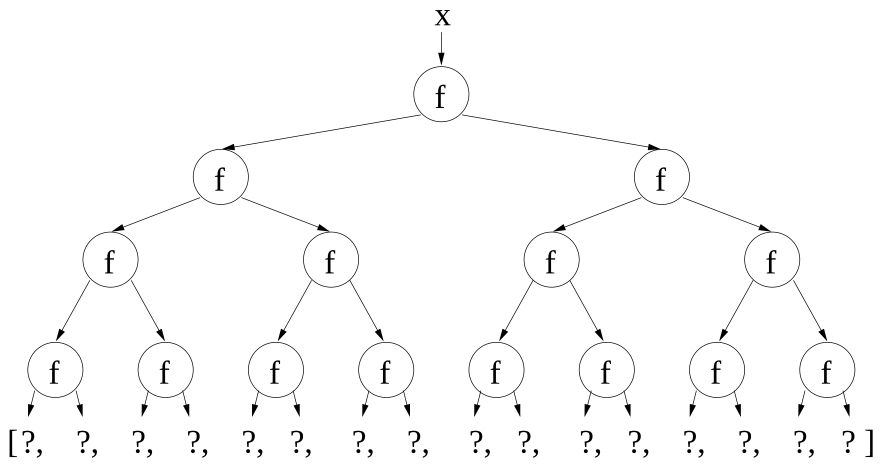


a divide with no conquer

# Conquer




# Divide



# The base model

- whether to have a split model (SCL style)
- imperative v. declarative v. explicitly parallel ?
- my (pragmatic? ) preference
  - split, with imperative sequential as a first step
  - split, with imperative parallel as a goal

# Cost Calculus

- to aid the design/derivation process
- compositionality
- underlying cost/computational model
  - realistic enough to be genuinely reflective (at least in relative of not absolute terms)
  - simple enough to be portable across varying architectures
  - practical enough to be competitive with ‘direct’ programming
  - **able to reflect and reward higher level computational structure?**

- PRAM
  - tractable but unrealistic (?)
  - uncompetitive
- BSP
  - realistic, simple, practical
  - ‘bulk’ property attractive
  - no structure (fundamentally so?)
- H-PRAM (Hierarchical)
  - same drawbacks as PRAM
  - good structure

- the ideal?
  - BSP-like parameter set
  - respecting locality
  - respecting interaction structure (i.e. not just arbitrary ‘h-relations’)
- are these irretrievably conflicting goals?
- is it better to join the winning bandwagon?

# Formal Methods

- a “Good Thing”
- pragmatic target
  - derivation/transformation at skeletal level
  - particularly algorithmic transformations (eg d&c to pipe)
  - trust for bulk of sequential base across well defined interface
- schemes/identities which are of particular interest when the underlying framework is parallel (homomorphisms etc)

# Doing It

- need real implementations
  - second order control across processor groups
- need to engage with ‘real’ parallel programmers
- need to find out what the pragmatically useful structures are
- need to compromise on language models (up to a point)

# Where do we go now?

- more meetings?
- ‘grand challenges’?
- loosely shared agenda?

**Let’s talk about it!**