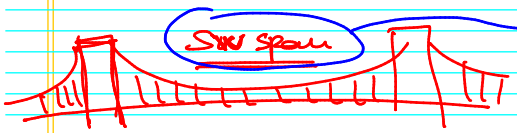
A View of the parallel computing landscape

CACM OCT'09



hw

appl

Amdhal

100 cores = 90 small cores + 1 "big" core

10% → sp = 2

s(n) ~ 9.2 ~ 17

3 challenges

→ seq programmers (need for)

→ compiler / os
↳ need time to be adapted to //

→ how to measure tools / frameworks improvements

APPL tower

↳ 1) kind of applications

2) criteria used for evaluating
performance, worst response time, battery life, reliability, security

PARLAB

APPL — MUSIC/HEARING — PARALLEL BROWSER
SPEAK RECOGNITION — NAME WHISPER
MEDICAL — HEATH COACH

Software SPAN
- patterns (parallel) vs programming languages
- split productivity & eff layers
- autotuning vs compilers
- synthesis with sketching
- efficiency, energy --
- verification & testing, not one or the other
- space/time partitioning for decomposed OSs

Answer: Intuitive performance models
Acarate, complete counters perf & energy

Reasons for optimism
→ there is no killer uprocess
→ all the wood behind one arrow
→ FPGAs prototypes → shorten the hw/sw cycle

TBB (Intel) TPL (Microsoft)

multicores → ... → we need threads → High level tools for threading

TBB thread building blocks library

2005 → 3.0

- parallel loops
  - parallel_for (range, body)
  - parallel_do
- reduce
- pipeline
  - create
  - add-stages
  - start
- for containers
- tasks & scheduling (spawn)

TPL task parallel library

Microsoft .NET framework

task
data
} parallelism

```
Parallel.ForEach(sourceColl, item => Body(item));
                              ↑x.Body(x)

Parallel.Invoke( () => Work1(), () => Work2() )
```

Pitfalls (TPL)

do not assume that parallel is always faster
avoid writing to shared memory locations
avoid over-parallelization
avoid to call non thread safe methods
limit calls to thread-safe methods
do not assume iterations of a per loop always go parallel