Parallel Programming by Hints

Chen Ding

University of Rochester

presented by Xiaoming Gu
Safe Parallel Programming

• Hints express likely rather than definite parallelism
  • bop ppr { code }
    • PPR means possibly parallel region
    • the PPR block may be parallel with the code after the PPR block
  • bop ordered { code }
    • the ordered block should be run one task at a time and in the sequential order

• Sequential equivalence
  • a parallel execution is allowed if its results are the same as sequential execution
  • incorrect hints may lose parallelism but won’t affect result
  • no non-determinism, no dead/live lock, no parallel debugging
Parallelism and Dependence

- No dependence
  - embarrassingly parallel
- False dependence
  - remove by data copy-on-write in bop ppr
  - sequential merging
- True dependence
  - serialization through bop ordered [Ke et al. OOPSLA 2011]
- Recent uses of copy-on-write
  - speculative parallelization
    - BOP, PLDI 2007; CorD/Spice C, MICRO’08, PLDI’10, PPOPP’11; SMTX, ASPLOS’09
  - race-free and deterministic execution of threaded code
    - Grace, OOPSLA’09; CoreDet, ASPLOS’10; Determinator, OSDI’10; DoublePlay, ASPLOS’11.
A program execution is a series of PPRs

Copy-on-write in each PPR

Sequential commit as PPRs finish

Recovery by understudy
  • in case speculation wrong or too slow
Example One

- parallel if \( x \neq y \)
- ppr is fork-w/o-join
- copy-on-write so concurrent execution does no harm to ppr1
- sequential commit to check \( x,y \) to ensure no conflict
- understudy in case ppr2 is wrong or too slow
- need to monitor \( g[x] \) and \( g[y] \)
- automatic through page protection [PLDI’07]
- manual annotation (next slide)

```plaintext
# try foo in parallel
bop_ppr {  
g[x] = foo(x) 
}

# try bar in parallel
bop_ppr {  
  bar(g[y]) 
}
```
**Access Monitoring via Annotation**

- `bop_promise( g[x] )` marks a write
  - the last value is exposed at the end
  - otherwise invisible
- `bop_use( g[y] )` marks a use
  - guaranteed to be same as in sequential execution
- the actual parameters in C
  - `bop_promise/use(&g[x], sizeof(g[x]))`

```c
# try foo in parallel
bop_ppr {
    bop_promise( g[x] )
    g[x] = foo( x )
}
```

```c
# try bar in parallel
bop_ppr {
    bop_use( g[y] )
    bar( g[y] )
}
```
• Example 2: `strcmp`
  • part of libc
• Uncertain parallelism
  • length of the string is unknown
  • location of the first difference unknown

```c
while (!done) {
    bop_use( done )
    ptr1, ptr2 = str1, str2

    bop_ppr {
        do {
            c1 = *ptr1++
            c2 = *ptr2++
        } while (c1==c2 && ptr1<base1+step)
        if ( c1==’\0’ || c1!=c2 ) {
            ret = c1 – c2
            done = true
            bop.promise( ret )
            bop.promise( done )
        }
    }
}
str1 += step
str2 += step
}
return ret
```
Why Programming by Hints

- High-level language
  - pioneered by Fortran
  - letting the compiler to write machine code
  - retaining good performance
- For high-level language to suggestion language
  - HLL: hides details of a machine
  - SL: hides details of a program
- Benefits
  - productivity, composability, portability, incremental

- For more information
  - Ke et al. OOPSLA 2011, Ding et al. PLDI 2007