Classical Paradigms in concurrency

Frédéric Haziza <daz@it.uu.se>

Department of Computer Systems
Uppsala University

Summer 2010
Classical Paradigms

- Trivial parallelism
- Data parallelism
- Task parallelism / Functional parallelism

5 paradigms:
- Iterative parallelism
- Recursive parallelism
- Producer/Consumer
- Client/Server
- Interacting peers
Iterative Parallelism: Matrix multiplication

1:   double a[n,n], b[n,n], c[n,n];
2:   for [ i=0 to n-1 ] {
    ‣ iterating through the rows
3:       for [ j=0 to n-1 ] {
          ‣ iterating through the columns
4:             ‣ Computes inner product of a[i,*] and b[*j]
5:             c[i,j] = 0.0;
6:       for [ k = 0 to n-1 ] {
7:         c[i,j] = c[i,j] + a[i,k]*b[k,j];
8:       }
9:   }
10: }

What can we parallelize? Line 5 to 8
⇒ c[i,j] is written to, and a[i,k], b[k,j] are only read
⇒ every c[i,j] computation!
Iterative Parallelism: Matrix multiplication

Parallelizing the rows

CONC [ i=0 to n-1 ] {
    compute rows in parallel
    for [ j=0 to n-1 ] {
        c[i,j] = 0.0;
        for [ k = 0 to n-1 ] {
            c[i,j] = c[i,j] + a[i,k]*b[k,j];
        }
    }
}
Iterative Parallelism: Matrix multiplication

Parallelizing the columns

\[ \text{CONC} \ [ j=0 \ \text{to} \ n-1 ] \ { \ \text{\textit{compute columns in parallel}} \}
\]
\[ \text{for} \ [ i=0 \ \text{to} \ n-1 ] \ { \} \]
\[ c[i,j] = 0.0; \]
\[ \text{for} \ [ k = 0 \ \text{to} \ n-1 ] \ { \} \]
\[ c[i,j] = c[i,j] + a[i,k]*b[k,j]; \]
\[ } \]
\[ } \]
\[ } \]
Iterative Parallelism: Matrix multiplication

Parallelizing all rows and columns

CONC [ i=0 to n-1, j=0 to n-1 ] {
    c[i,j] = 0.0;
    for [ k = 0 to n-1 ] {
        c[i,j] = c[i,j] + a[i,k] * b[k,j];
    }
}
Recursive Parallelism: Adaptive Quadrature

\[ \int_{a}^{b} f(x) \, dx \]
Recursive Parallelism: Adaptive Quadrature

1: double fleft = f(a), fright, area = 0.0;
2: double width = (b-a)/ INTERVALS;

3: for [ x = (a+width) to b by width ] {
4:   fright = f(x);
5:   ▶ Compute the small rectangle area
6:   area = area + (fleft + fright) * width / 2;
7:   fleft = fright; ▶ the right-hand value becomes the new left-hand value
8: }

\[ y \]
\[ f(x) \]
\[ x \]
Divide and Conquer

\[ f(x) \]

\[ \left| \text{area}_{\text{new}} - \text{area}_{\text{old}} \right| > \epsilon \]
Divide and Conquer

```c
double quad(double left, right, fleft, fright, oldarea) {
    double mid = (left + right)/2; ▷ find the middle point
    double fmid = f(mid); ▷ get its value
    double larea = (fleft + fmid) * (mid - left)/2;
    double rarea = (fmid + fright) * (right - mid)/2;

    if |(larea + rarea) - oldarea| > ϵ {
        ▷ Recurse to integrate both halves
        larea = quad(left, mid, fleft, fmid, larea);
        rarea = quad(mid, right, fmid, fright, rarea);
    }
    return (larea + rarea);
}
```

\[
\int_{a}^{b} f(x)\,dx \approx quad(a, b, f(a), f(b), (f(a) + f(b)) \times (b - a)/2);
\]
double quad(double left, right, fleft, fright, oldarea) {
    double mid = (left + right)/2;  // find the middle point
    double fmid = f(mid);           // get its value
    double larea = (fleft + fmid) * (mid - left)/2;
    double rarea = (fmid + fright) * (right - mid)/2;

    if |(larea + rarea) - oldarea| > \epsilon 
        // Recurse to integrate both halves
        CONC [ ] {
            larea = quad(left, mid, fleft, fmid, larea);
                // in parallel!
            rarea = quad(mid, right, fmid, fright, rarea);
        }  // Must wait for larea and rarea
    }
    return (larea + rarea);
}
Producer / Consumer
Client / Server

- Client\textsubscript{n}
- Server
- Request
- Reply

\[Client\textsubscript{1} \rightarrow \cdots \rightarrow Client\textsubscript{n} \rightarrow Server \rightarrow \cdots \rightarrow \text{Reply} \rightarrow \text{Request}\]

OS2’10 | Classical Paradigms (in concurrency)
Interacting Peers - Coordinator/Workers

- Coordinator
- Worker\(_1\)
- \(\text{Worker}_{n-1}\)
- Results
- Data
Interacting Peers - Circular Pipeline

Worker_1 → ... → Worker_{n-1}
Interacting Peers

Coordinator/Workers

Worker₁

Results

Data

Workerₙ₋₁

Results

Data

Circular pipeline

Worker₁

... → Workerₙ₋₁