Classical Paradigms
in concurrency

Frédéric Haziza <daz@it.uu.se>

Department of Computer Systems
Uppsala University

Summer 2009

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 trough the rows
3:      for j=0 to n-1  {    ▷iterating trough 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];
            }
        }
    }
```

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

```
CO [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

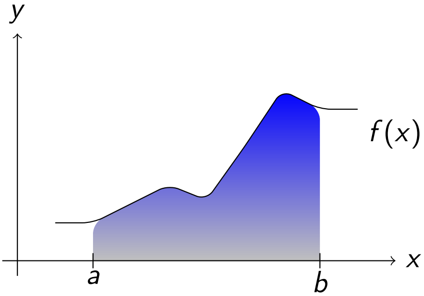
```
CO [j=0 to n-1]  {    ▷compute columns in parallel
    for i=0 to n-1  {
        c[i,j] = 0.0;
        for  k = 0 to n-1   {
            c[i,j] = c[i,j] + a[i,k]*b[k,j];
        }
    }
}
```

Iterative Parallelism: Matrix multiplication

Parallelizing all rows and columns

```
CO [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

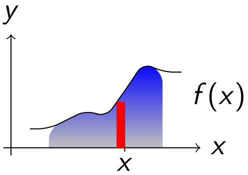


$$\int_a^b f(x)\mathrm{d}x$$

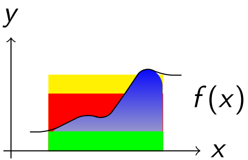
Recursive Parallelism: Adaptive Quadrature

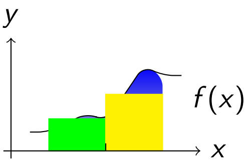
```
double fleft = f(a), fright, area = 0.0;
double width = (b-a)/ INTERVALS;

for x = (a+width) to b by width {
    fright = f(x);
     ▷Compute the small rectangle area
    area = area + (fleft + lfright) * width / 2;
    fleft = fright;  ▷the right-hand value becomes the new left-hand value
}
```



Divide and Conquer





$$|area_{new} - area_{old}| > \epsilon$$

Divide and Conquer

```
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);
}
```

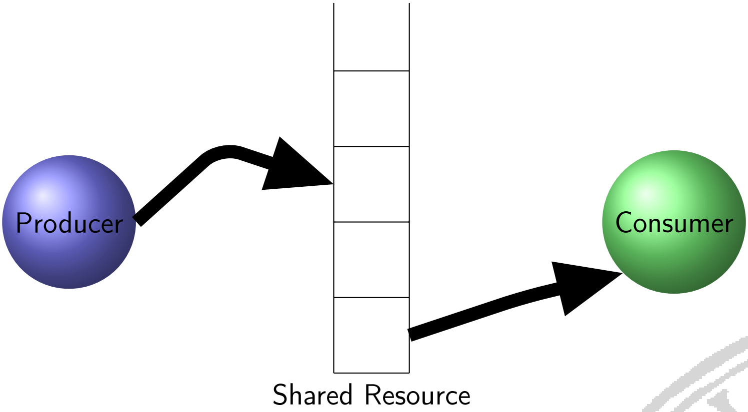
$$\int_a^b f(x)\mathrm{d}x \approx quad(a, b, f(a), f(b), (f(a) + f(b)) * (b - a)/2);$$

Divide and Conquer - Parallel

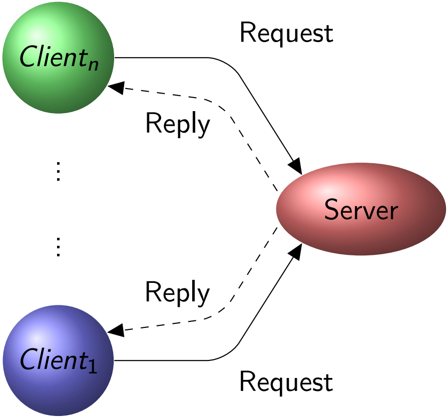
```
double quad(double left, right, fleft, fright, oldarea) {
    double mid = (left + right)/2;   ▷find the middle point
    double fmid = f(mid);            ▷get its value
    double larea = (fleft + fmid) * (mid − left)/2;
    double rarea = (fmid + fright) * (right − mid)/2;

    if |(larea + rarea) − oldarea| > ε {
          ▷Recurse to integrate both halves
        CO [] {
            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);
}
```

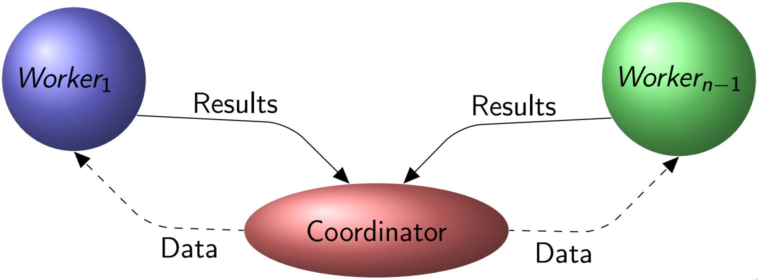
Producer / Consumer



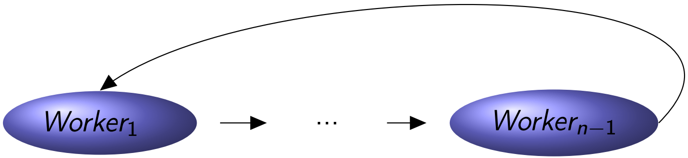
Client / Server



Interacting Peers - Coordinator/Workers

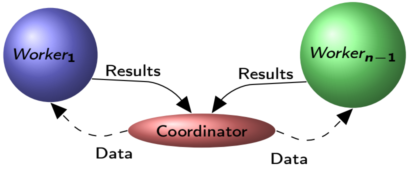


Interacting Peers - Circular Pipeline



Interacting Peers

Coordinator/Workers



Circular pipeline