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]; ▷
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

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

Parallelizing the columns

```cpp
CO [j=0 to n-1] {
    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

\[
\text{CO } [i=0 \text{ to } n-1, j=0 \text{ to } n-1] \quad \{
\begin{align*}
  c[i,j] &= 0.0; \\
  \text{for } &k = 0 \text{ to } n-1 \quad \{
  \quad c[i,j] &= c[i,j] + a[i,k] \times b[k,j]; \\
  \quad \}
  \\
\end{align*}
\}
\]
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: \textbf{for} x = (a+width) to b by width \{ 
4: \quad fright = f(x);
5: \quad \triangleright Compute the small rectangle area
6: \quad area = area + (fleft + fright) * width / 2;
7: \quad fleft = fright; \quad \triangleright the right-hand value becomes the new left-hand value
\}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{quadrature.png}
\caption{Approximating the area under the curve $f(x)$ using adaptive quadrature.}
\end{figure}
Divide and Conquer

\[ f(x) \]

\[ x \]

\[ |area_{new} - area_{old}| > \epsilon \]
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);
}

\[
\int_{a}^{b} f(x)dx \approx quad(a, b, f(a), f(b), (f(a) + f(b)) \times (b - a)/2);
\]
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| > 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

Shared Resource

Producer

Consumer
Client / Server

Request

\( Client_n \)

\( Client_1 \)

Reply

Server

OS2'09 | Classical Paradigms (in concurrency)
Interacting Peers - Coordinator/Workers

Worker\(_1\)  

Results \rightarrow Coordinator \rightarrow Results  

Data \rightarrow Coordinator \rightarrow Data  

Worker\(_{n-1}\)
Interacting Peers - Circular Pipeline

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

 Coordinator/Workers

 Circular pipeline

Worker_1

Data

Results

Coordinator

Worker_{n-1}

Data

Results

Worker_1

...  

Worker_{n-1}