Expressing Parallel Computation

Arvind
Laboratory for Computer Science
M.I.T.

Main Stream Parallel Computing

PC class SMPs are about to cause a revolution

- cheap
- run NT, Solaris, Linux
- will track technology

⇒

- *sold in large numbers (100,000)*
- *incentive for ISVs to produce parallel software*
  compilers, debuggers, performance tools
  applications, libraries, ...

Deluxe SMPs

Deluxe SMPs have a superior memory system than low-end SMPs but cost more

64-Processor SMP >> 16 x 4-Processor SMPs

- **SUN Enterprise Series:**
  - **E5000 up to 12 processors**
  - **E10000 up to 64 processors**
- **SGI**
  - **Origin 2000 32 to 128 processors**
- **Digital**
  - **Turbo Laser 8400 up to 12 processors**
  - **Raw Hide 4 processors**

IBM, NEC, Hitachi, Fujitsu, ...

Very large scaleable parallel machines are built as clusters of SMPs

SMP Market

Current market : "servers"

SMPs are poised to replace all mainframes **$$$$**

databases
OLTP
Web servers
Internet commerce ...

Potential Market

SMPs can take over all Technical/Scientific computing **$**

CAD
CAM
weather, climate...
drug design ...

Parallel machines today, however, run primarily sequential programs !!!

Why a lack of Parallel Applications

Applications follow cost-effective hardware which has become available only since 1996

Technical reasons:

- **automatic parallelization is difficult**
- **parallel programming is difficult**
- **parallel programs run poorly on sequential machines**

⇒ need to lower the *entry cost of parallel programming*

Why not use sequential languages ?

Algorithm with parallelism

↓ ***encode***

Program with sequential semantics

↓ ***detect parallelism***

Parallel code **!!!**

Matrix Multiply

$C = A \times B$ $\qquad\qquad$ $C_{i,j} = \sum_k A_{i,k} B_{k,j}$

All $C_{i,j}$'s can be computed in parallel.
In fact, all multiplications can be done in parallel!

Fortran

```
do i = 1,n
 do j = 1,n
  do k = 1,n
    C(i,j) = C(i,j) + A(i,k)*B(k,j)
  continue
 continue
continue
```

Parallelism?

Detecting Parallelism

Data-dependence

x[i] = ...

= .... x[j].... $\qquad$ **?**

Anti-dependence

= .... x[i]....

x[j] = ... $\qquad$ **?**

- **Data dependence analysis has to be conservative.**
- **Very difficult in the presence of pointers and complex subscripts**

Parallelism in *Matrix Multiply*

C = A x B $\qquad C_{i,j} = \sum_k A_{i,k} B_{k,j}$

Fortran

```
do i = 1,n
 do j = 1,n
  s = 0
   do k = 1,n
     s = s + A(i,k)*B(k,j)
   continue
  C(i,j) = s
 continue
continue
```

- **Shared `s` prevents parallel execution!**
- **An optimizing compiler can transform the program to enhance parallelism**

Parallelizing Compilers

More than 25 years of intensive research

- **only *limited success***
  - **instruction-level parallelism at the *basic-block* level**
  - **parallelism in *nested for-loops* containing arrays with simple index expressions**
- **instead of training compilers to recognize parallelism, *people have been trained* to write programs that parallelize.**

The research goes on ...

newer ideas: *Commutativity Analysis* of Object-oriented programs

⇒ *new programming constructs, languages, models*

Parallel Programming Models

High-level

Data parallel: *Fortran 90, HPF, ...*

Multithreaded: *Cid, Cilk,...*
Id, pH, Sisal, ...

Low-level

Message passing: *PVM, MPI, ...*

Threads & synchronization:
Futures, Forks, Semaphores, ...

Properties of Models

Determinacy - is the behavior of a program repeatable?

Compositionality - can independently created subprograms be combined in a meaningful way?

Expressiveness - can all sorts of parallelism be expressed?

Implementability - can a compiler generate efficient code for a variety of architectures?

Safer Ways of Expressing Parallelism

Data parallel: Fortran 90, HPF, ...
sources of parallelism - vector operators, *forall*
communication is specified by shift operations.

Implicit Parallel Programming: Id, pH, Sisal, ...
functional and logic languages specify
only a partial order on operations.

Determinacy of programs is guaranteed
⇒ *easier debugging !!!*

Programming is independent of machine configuration.

Data Parallel Programming Model

- All data structures are assigned to a grid of virtual processors.
- Generally the owner processor computes the data elements assigned to it.
- *Global communication* primitives allow processors to exchange data.
- *Implicit global barrier* after each communication.
- All processors execute the *same program* .

↓
communicate
(global)
↓
compute
(local)
↓
communicate
(global)
↓
compute
(local)
↓

Data Parallel *Matrix Multiply*

```
Real Array(N,N) :: A, B, C

Layout A(:NEWS, :NEWS), B(:NEWS, :NEWS)
       C(:NEWS, :NEWS)   ← each element is on a virtual
                           processor of a 2D grid
```

Set up the initial distribution of A and B

```
Do i = 1,N-1
 !Shift rows left and columns up
  A = CShift(A, Dim=2, Shift=1)  ← communication
  B = CShift(B, Dim=1, Shift=1)  ← communication
  C = C + A * B    ← parallel operations
End Do
```


Data Parallel Model

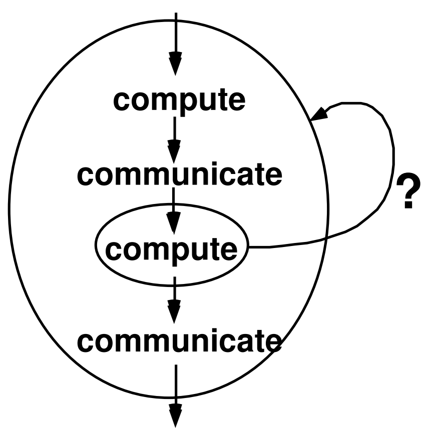
+ Good implementations are available

- Difficult to write programs

+ Easy *to debug* programs because of a single thread

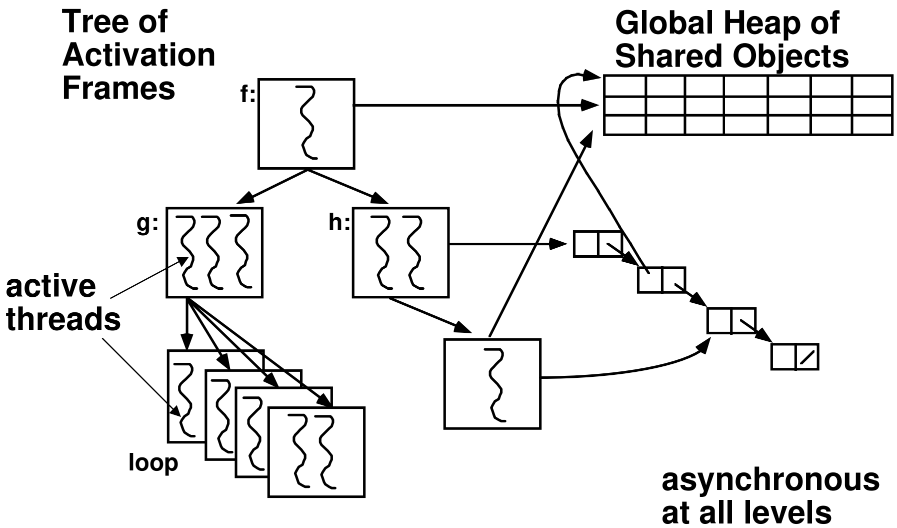
+ Implicit synchronization and communication

- Limited *compositionality!*



For *general-purpose programming*, which has more *unstructured parallelism*, we need more flexibility in scheduling.

Fully Parallel, Multithreaded Model



Explicit vs Implicit Multithreading

Explicit:

C + forks + joins + locks

multithreaded C: Cid, Cilk, ...

easy path for exploiting coarse-grain parallelism in existing codes

error-prone if locks are used

Implicit:

languages that specify only *a partial order on operations*

functional and *logic* languages: Id, pH, ...

safe, high-level, but difficult to implement efficiently without shared memory & ...

Explicitly Parallel *Matrix Multiply in Cilk*

```
cilk void
matrix_multiply(int** A, int** B, int** C,
                                        int n)
{     ...
      ...
      for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
           C[i][j] = spawn IP(A, B, i, j, n);

      sync;
      ...
      ...
}

int IP( ... ) { ... }
```


Implicitly Parallel *Matrix Multiply in pH*

```
make_matrix ((1,1),(n,n)) f
```

where `f` is the filling function

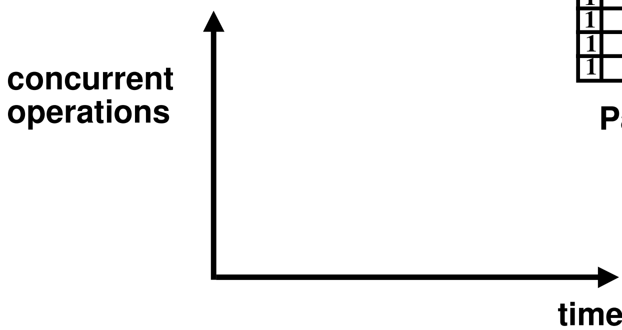
```
\(i,j).(IP (row i A) (column j B))
```

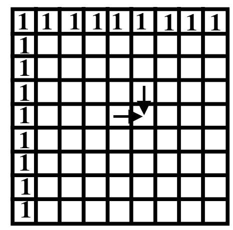
`make_matrix` ***does not specify the order in which the matrix is to be filled!***

no implication regarding the distribution of computation and data over a machine.

The *Wavefront* Example

$$x_{i,j} = x_{i-1,j} + x_{i,j-1}$$



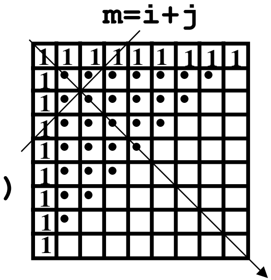


Parallelism?

Exposing Parallelism by Annotations

Wavefront **using *doalls* and *barriers***

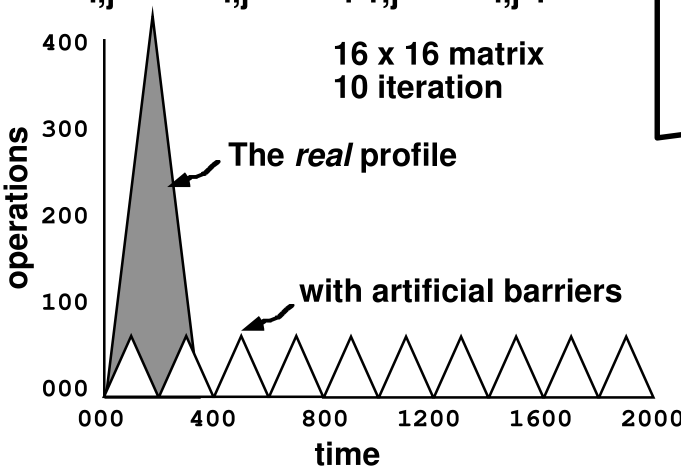
```
C  Initialize the top row and left column
       . . .
    barrier
C  Fill the upper left triangle
    do m = 4, n+1
     doall i = 2, m-2
       j = m-i
       A(i,j) = A(i-1,j) + A(i,j-1)
      continue
     barrier
    continue

C  Fill the lower right triangle
       . . .
```

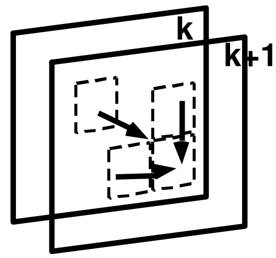


Ad hoc & Unsafe !

Repeated Wavefront (SOR)

$$x^{k+1}_{i,j} = x^{k}_{i,j} + x^{k+1}_{i-1,j} + x^{k+1}_{i,j-1}$$





Producer - Consumer parallelism is difficult to express using annotations

Implicitly Parallel Languages

Expose all types of parallelism, permit very high level programming

but

may be difficult to implement efficiently

Sometimes it is difficult to identify the source of inefficiency

the language?	the compiler?
the run-time system?	the architecture?

Future

Id pH **Cilk HPF**

multithreaded intermediate language

notebooks SMPs MPPs

Freshman in will be taught sequential programming as a special case of parallel programming in future

6.827
Multithreaded Languages and Compilers

I Issues in parallel programming — **5 lectures**
- *Explicit vs implicit parallel programming*
- *Memory model*
- *Scheduling*

II Fundamentals of functional programming — **8 lectures**
- *The λ-calculus, evaluation strategies*
- *Types, Polymorphism, Overloading*
- *Lists, Arrays, ...*
- *Monads, barriers*

III Non-Determinism — **5 lectures**
- *The λ_S-calculus*
- *I-structures and M-structures*

IV Multithreaded Compiling — **6 lectures**
- *SMT abstract machine*
- *run-time system*
- *strictness and partitioning analysis*

V Miscellaneous topics — **2 lectures**
- *Memory Models*
- *Static analysis*

...