Expressing Parallel Computation

Arvind
Laboratory for Computer Science
M.I.T.

Lecture 1

http://www.csg.lcs.mit.edu/6.827
Main Stream Parallel Computing

• Most “server” class machines these days are symmetric multiprocessors (SMP’s)
  – PC class SMP’s
    • 2 to 4 processors
    • cheap
    • run Windows & Linux
    • track technology
  – Delux SMP’s
    • 8 to 64 processors
    • expensive (16-way SMP costs >> 4 x 4-way SMPs)

• Applications
  – databases, OLTP, Web servers, Internet commerce...
  – potential applications: EDA tools, technical computing,...

http://www.csg.lcs.mit.edu/6.827
Large Scale Parallel Computing

• Most parallel machines with hundreds of processors are built as clusters of SMP’s
  – usually custom built with government funding
  – expensive: $10M to $100M
  – are treated as a national or international resource
  – Total sales are a tiny fraction of “server” sales
  – hard time tracking technology

• Applications
  – weather and climate modeling
  – drug design
  – code breaking (NSA, CIA, …)
  – basic scientific research
  – weapons development

Few independent software developers;
Programmed by very smart people

http://www.csg.lcs.mit.edu/6.827
Paucity of Parallel Applications

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

- Important applications are hand crafted (usually from their sequential versions) to run on parallel machines
  - explicit, coarse-grain multithreading on SMP’s
    - most business applications
  - explicit, coarse-grain message passing on large clusters
    - most technical/scientific applications

- Technical reasons:
  - automatic parallelization is difficult
  - parallel programming is difficult
  - parallel programs run poorly on sequential machines

Can the entry cost of parallel programming be lowered?

http://www.csg.lcs.mit.edu/6.827
Why not use sequential languages?

Algorithm with parallelism → encode 
Program with sequential semantics → detect parallelism 
Parallel code

http://www.csg.lcs.mit.edu/6.827
Matrix Multiply

\[ C = A \times B \]

\[ C_{i,j} = \sum_k A_{i,k} B_{k,j} \]

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
            s = s + A(i,k)*B(k,j)
            continue
        C(i,j) = s
        continue
    continue
```

Parallelism?

http://www.csg.lcs.mit.edu/6.827
Parallelizing Compilers

After 30 years of intensive research

• only limited success in parallelism detection and program transformations
  – instruction-level parallelism at the basic-block level can be detected
  – parallelism in nested for-loops containing arrays with simple index expressions can be analyzed
  – analysis techniques, such as data dependence analysis, pointer analysis, flow sensitive analysis, abstract interpretation, ... when applied across procedure boundaries often take far too long and tend to be fragile, i.e., can break down after small changes in the program.

• instead of training compilers to recognize parallelism, people have been trained to write programs that parallelize

http://www.csg.lcs.mit.edu/6.827
Parallel Programming Models

If parallelism can’t be detected in sequential programs automatically then design new 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, ...*

http://www.csg.lcs.mit.edu/6.827
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?

http://www.csg.lcs.mit.edu/6.827
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.

http://www.csg.lcs.mit.edu/6.827
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*.

http://www.csg.lcs.mit.edu/6.827
Data Parallel Matrix Multiply

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

Layout A(:NEWS, :NEWS), B(:NEWS, :NEWS)
C(:NEWS, :NEWS)

... set up the initial distribution of data elements ...

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

http://www.csg.lcs.mit.edu/6.827
+ 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.

http://www.csg.lcs.mit.edu/6.827
Fully Parallel, Multithreaded Model

Tree of Activation Frames

Global Heap of Shared Objects

active threads

loop

asynchronous at all levels

http://www.csg.lcs.mit.edu/6.827
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 languages*: Id, pH, ...
- *safe, high-level, but difficult to implement efficiently without shared memory & ...*

http://www.csg.lcs.mit.edu/6.827
Explicitly Parallel *Matrix Multiply*

```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(...)
```

Cilk program

http://www.csg.lcs.mit.edu/6.827
Implicitly Parallel Matrix Multiply

\[ \text{make}_{-}\text{matrix} \ (((1,1),(n,n)))\ f \]

where \( f \) is the filling function

\[ (i,j). (\text{IP (row} \ i \ \text{A) (column} \ j \ \text{B})) \]

\text{make}_{-}\text{matrix} \ does \ not \ specify \ the \ order \ in \ which \ the \ matrix \ is \ to \ be \ filled!} \]

\text{no \ implication \ regarding \ the \ distribution \ of \ computation \ and \ data \ over \ a \ machine.} \\

http://www.csg.lcs.mit.edu/6.827
Implicitly Parallel Languages

Expose all types of parallelism, permit very high level programming

*but*

may be difficult to implement efficiently

*Why?*
  - some inherent property of the language?
  - lack of maturity in the compiler?
  - the run-time system?
  - the architecture?

http://www.csg.lcs.mit.edu/6.827
Future

multithreaded intermediate language

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

http://www.csg.lcs.mit.edu/6.827
This subject is about

• implicit parallel programming in pH
• functional languages and the $\lambda$ calculus because they form the foundation of pH
• multithreaded implementation of pH aimed at SMP’s
• some miscellaneous topics from functional languages, e.g., abstract implementation, confluence, semantics, TRS...

This term I also plan to give 6 to 8 lectures on Bluespec, a new language for designing hardware. Bluespec also borrows heavily from functional languages but has a completely different execution model.

http://www.csg.lcs.mit.edu/6.827