A Hardware-design Inspired Methodology for Parallel Programming

Arvind
Computer Science & Artificial Intelligence Lab.
Massachusetts Institute of Technology

Workshop on Architectures and Compilers for Multithreading
December 14, 2007

Plan for this talk

- My old way of thinking (up to 1998)
  - “Where are my threads?”
  - Not necessarily wrong
- My new way of thinking (since mid 2006)
  - “Parallel program module as a resource”
  - Not necessarily right

Acknowledgement: Nirav Dave

Only reason for parallel programming used to be performance

- This made programming very difficult
  - Had to know a lot about the machine
  - Codes were not portable – endless performance tuning on each machine
  - Parallel libraries were not composable
  - Difficult to deal with heap structures and memory hierarchy
  - Synchronization costs were too high to exploit fine-grain parallelism

How to exploit 100s of threads from software?

Implicit Parallelism

- Extract parallelism from programs written in sequential languages
  - Lot of research over four decades – limited success
- Program in functional languages which may not obscure parallelism in an algorithm

If the algorithm has no parallelism then forget it

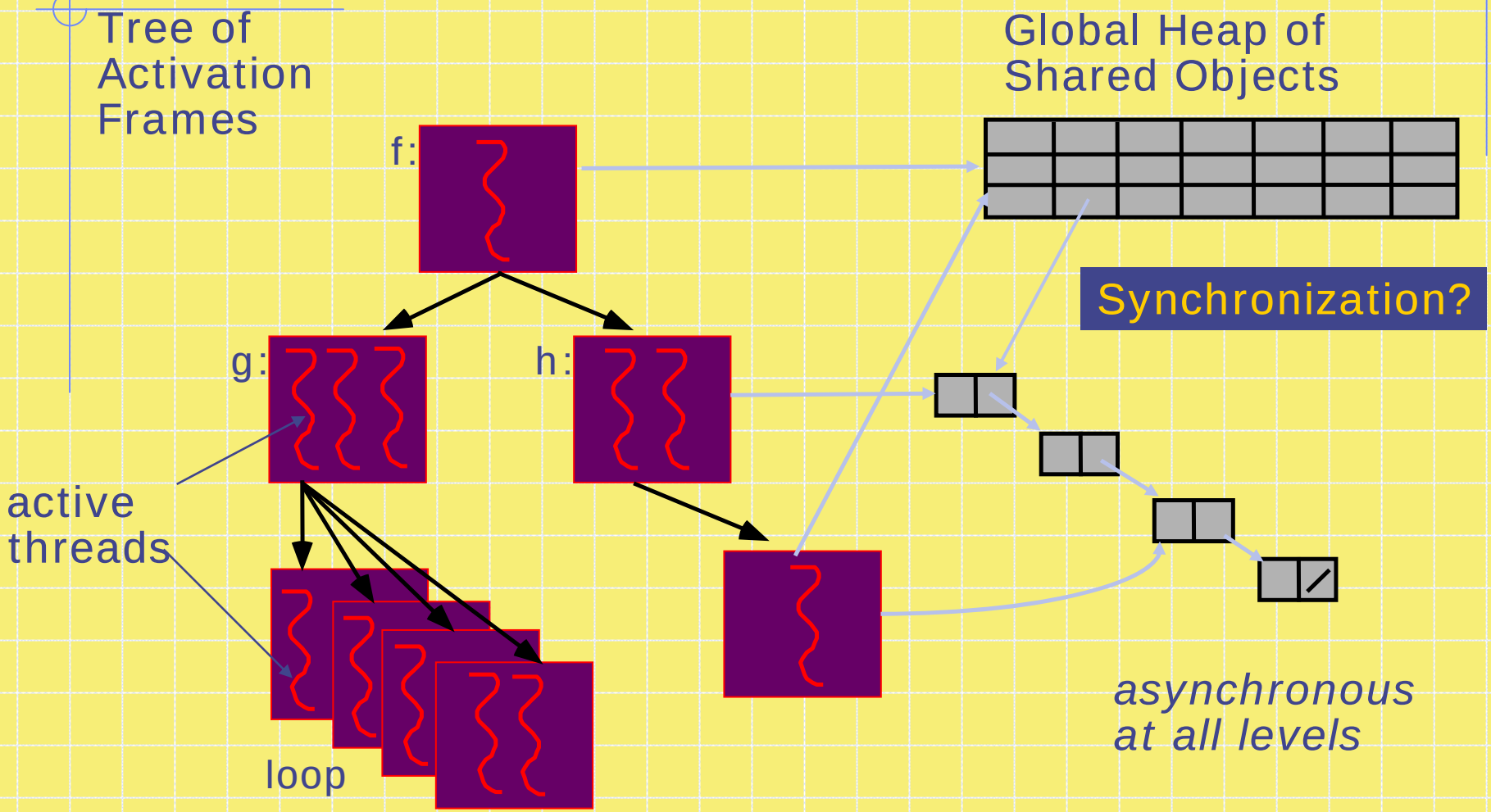
If parallelism can't be detected automatically …

Design/use new explicitly parallel programming models …

Works well but not general enough

- High-level
  - Data parallel: *Fortran 90, HPF, …*
  - Multithreaded: *Id, pH, Cilk,…, Java*
- Low-level
  - Message passing: *PVM, MPI, …*
  - Threads & synchronization:
    Forks & Joins, Locks, Futures, …

Fully Parallel, Multithreaded Model



Efficient mappings on architectures proved difficult

Functional Languages (Id) / Dataflow (Monsoon) Experience

- Monsoon was a simple, high performance design, easily exploited fine-grain parallelism, tolerated latencies efficiently
- Id preserved fine-grain parallelism which was abundant
- Robust compilation schemes; DFGs provided easy compilation target
- Issues:
  - No C or Fortran compiler for Monsoon
  - No Id or pH compiler for conventional parallel machines
  - Dataflow model gave you parallelism for free, but required analysis to get locality

My unrealized dream

A time when Freshmen will be taught sequential programming as a special case of parallel programming

Has the situation changed?

Multicores have arrived

- Functional Languages are going main stream
  - Google talks about map-reduce
  - Microsoft has released F#
- Explosion of cell phones
- Explosion of game boxes

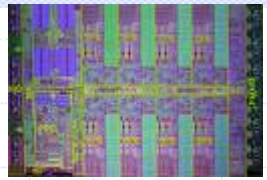
Freshmen are going to be hacking game boxes and cell phones

It is all about parallelism now!

SoC Trajectory:

multicores, heterogeneous, regular, …

IBM Cell Processor

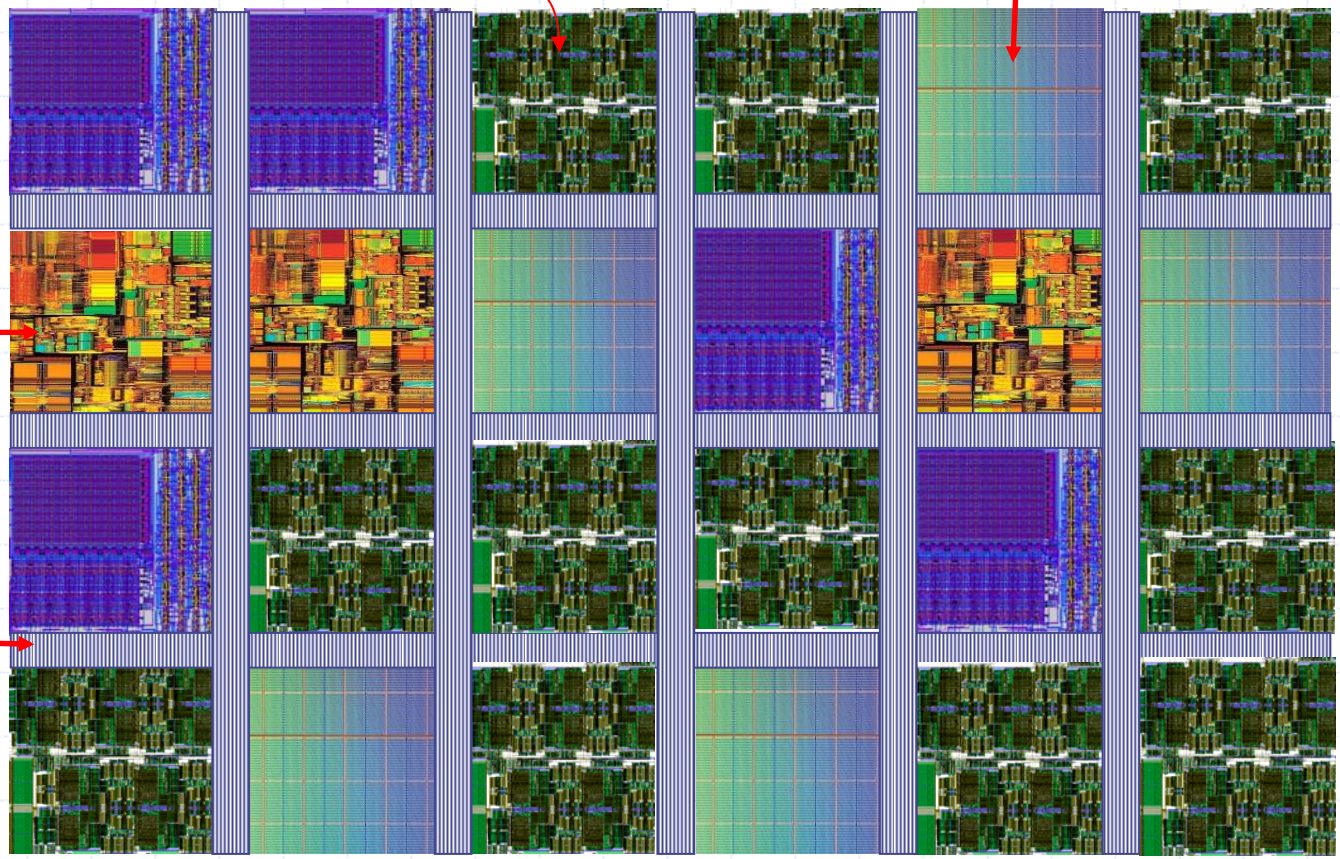


On-chip memory banks

Application-specific processing units

General-purpose processors

Structured on-chip networks



Can we rapidly produce high-quality chips and surrounding systems and software?

now ...

Cell phone

- Mine sometimes misses a call when I am surfing the web
  - To what extent the phone call software should be aware of web surfing software, or vice versa?
  - Is it merely a scheduling issue?
  - Is it a performance issue?

Sequential “modules” are often used in concurrent environments in unforeseen ways

New Goals

Synthesis as opposed to *Decomposition*

Know how to do this

- A method of designing and connecting modules such that the functionality and performance are predictable
  - Must facilitate natural descriptions of concurrent systems
- A method of refining individual modules into hardware or software for SoCs
- A method of mapping such designs onto "multicores"
  - Time multiplexing of resources complicates the problem

A hardware inspired methodology for "synthesizing" parallel programs

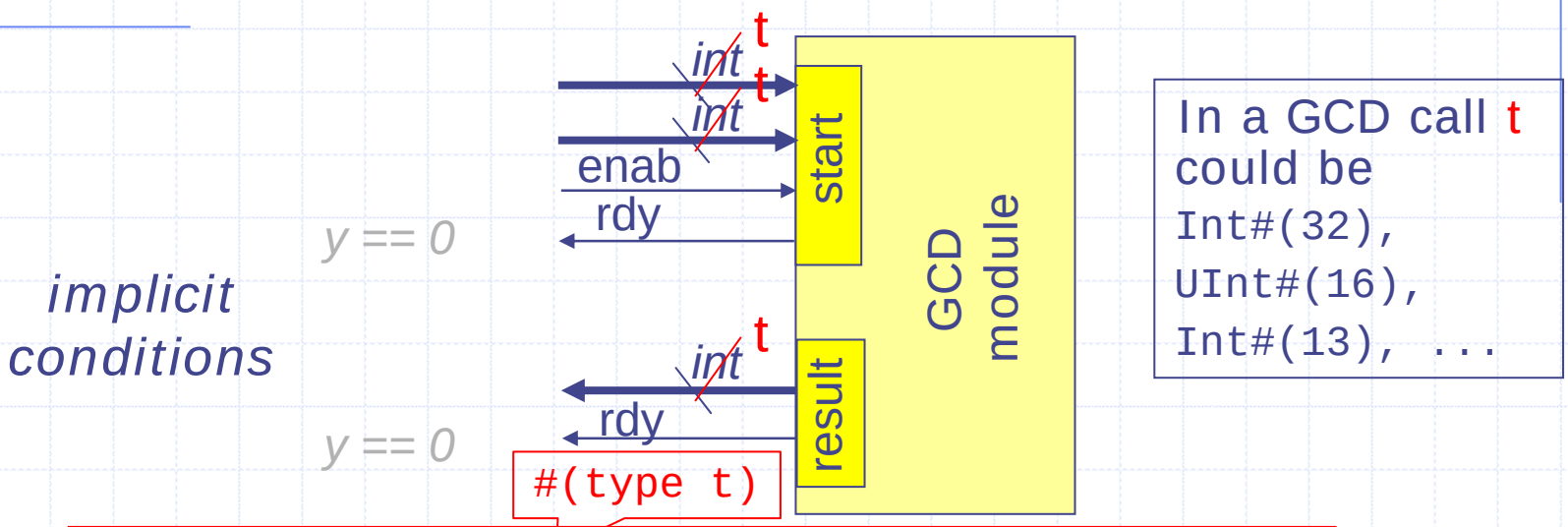
- Rule-based specification of behavior (Guarded Atomic Actions)
  - Lets you think one *rule* at a time
- Composition of modules with guarded interfaces

Bluespec

Unity – late 80s
Chandy & Misra

Closely connected with transactional memory

GCD Hardware Module



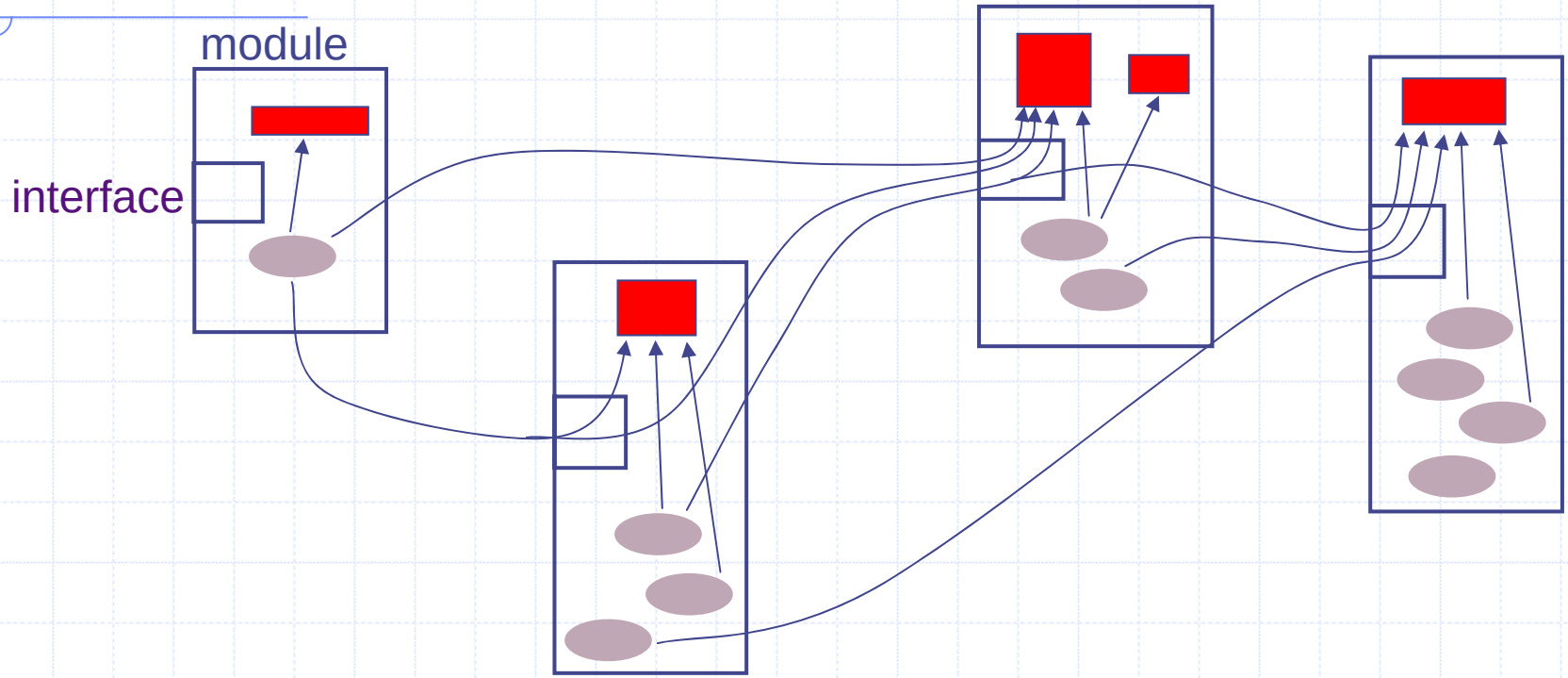
In a GCD call t could be
`Int#(32),`
`UInt#(16),`
`Int#(13), ...`

```
interface I_GCD;
    method Action start (int t a, int t b);
    method int t result();
endinterface
```

- The module can easily be made polymorphic
- Many different implementations, *including pure software ones*, can provide the same interface

```
module mkGCD (I_GCD)
```

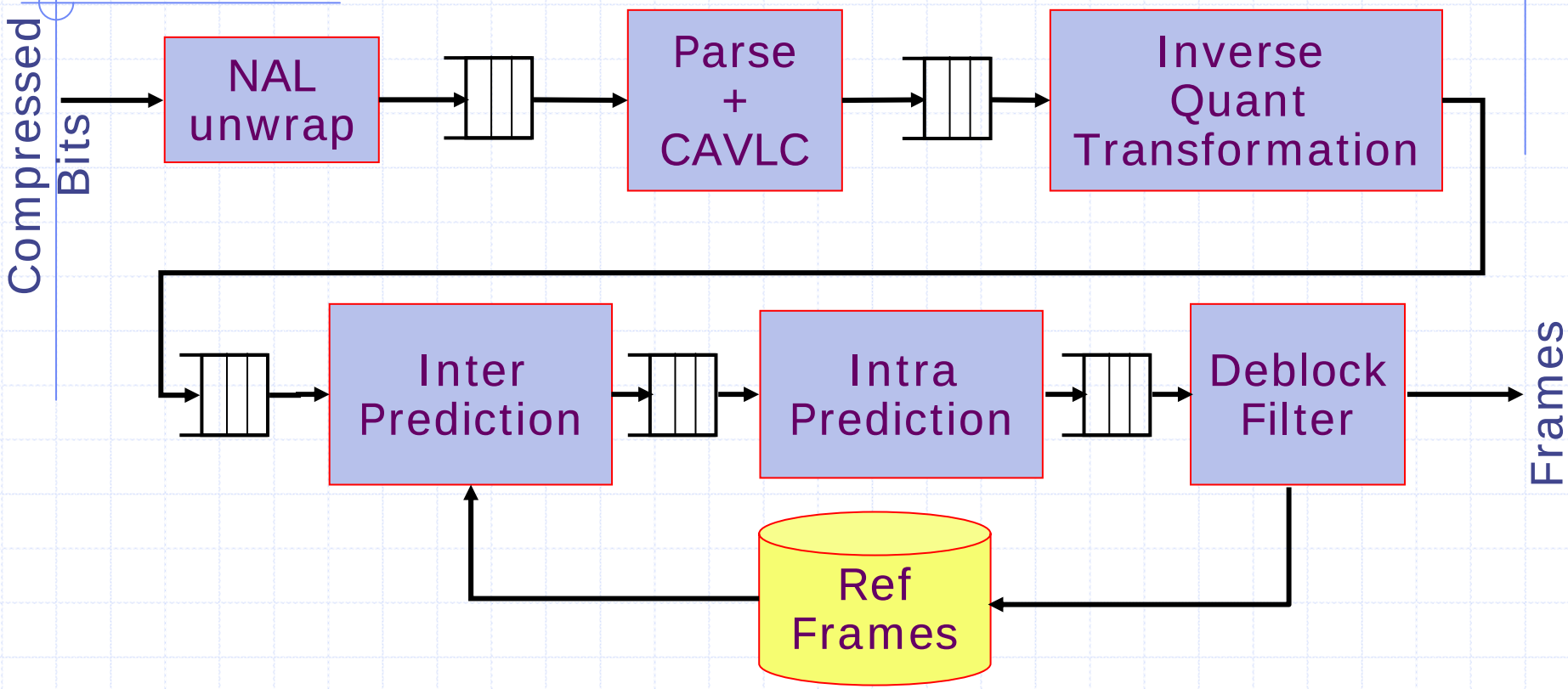
Bluespec: State and Rules organized into *modules*



- Each module embodies its own resources and is mapped on to its own hardware – *no time-multiplexing*

Parallel Programming can be easier than sequential programming

Example: H.264 Video Decoder



Different requirements for different environments

- QVGA 320x240p (30 fps)
- DVD 720x480p
- HD DVD 1280x720p (60-75 fps)

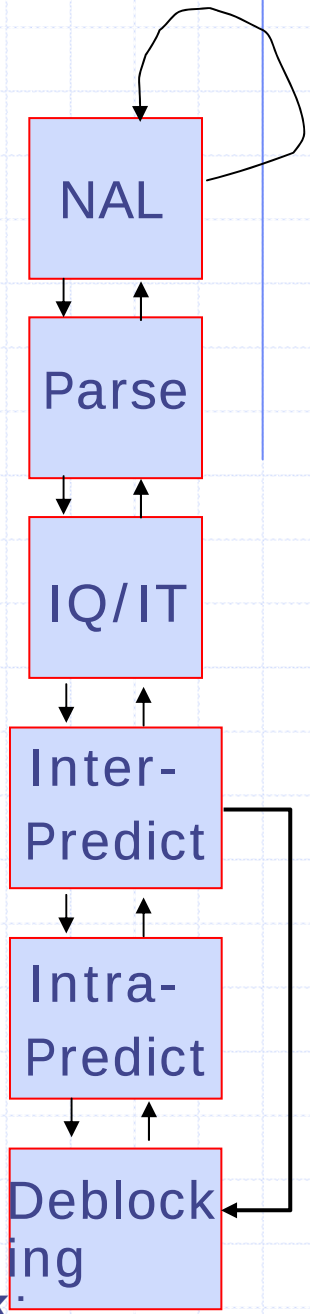
May be implemented in hardware or software depending upon ...

Sequential code

from ffmpeg

```
void h264decode(){
      int stage = S_NAL;
 while (!eof()){
  createdOutput = 0; stallFromInterPred = 0;
  case (stage){
    S_NAL: try_NAL();
        if (createdOutput) stage = S_Parse; break;
    S_Parse: try_Parse();
        stage=(createdOutput) ? S_IQIT: S_NAL; break;
    S_IQIT: try_IQIT();
        stage=(createdOutput) ? S_Parse:S_Inter; break;
    S_Inter: try_Inter();
        stage=(createdOutput) ? S_IQIT:S_Intra;
        if (stallFromInterPred) stage=S_Deblock; break;
    S_Intra: try_Intra();
        stage=(createdOutput) ? S_Inter:S_Deblock; break;
    S_Deblock: try_deblock(); stage= S_Intra; break } } }
```

20K Lines of C
out of 200K



Parallelizing the C code

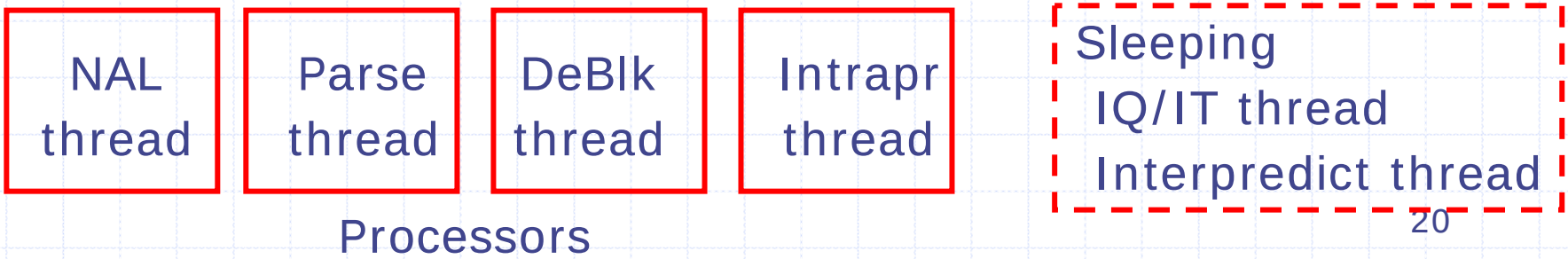
- Control structure is totally over specified and unscrambling it is beyond the capability of current compiler techniques
- Program structure is difficult to understand
- Packets are kept and modified in a global heap
- Thread-level data parallelism?

P Threads: *can be used to introduce different type of threads*

- A (p)thread of each block

```
int main(){
pthread_create(NAL);
phtread_create(Parse);
pthread_create(IQIT);
pthread_create(Interpred);
pthread_create(Intrapred);
pthread_create(Deblock);}
```

- But there is no control over mapping

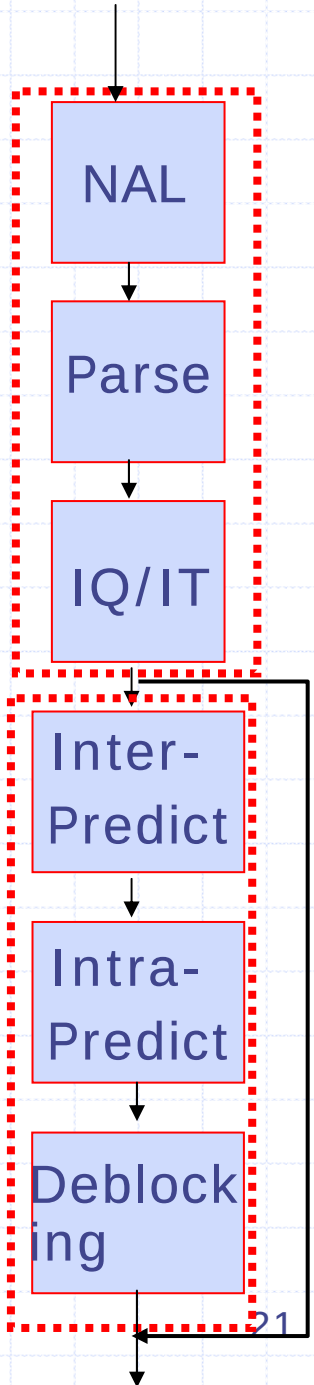


StreamIT

a more natural expression using filters

```
bit -> frame pipeline H264Decode {
    add; NAL();
    add; Parse();
    add; IQIT();
    add; feedbacloop{
        join roundrobin;
        body pipeline{
            add; InterPredict();
            add; IntraPredict();
            add; Deblock();}
split roundrobin;}}
```

Gives the required rates StreamIt compiler can do a great job of generating efficient code but not easy to express or compile feed-back



Functional languages (pH)

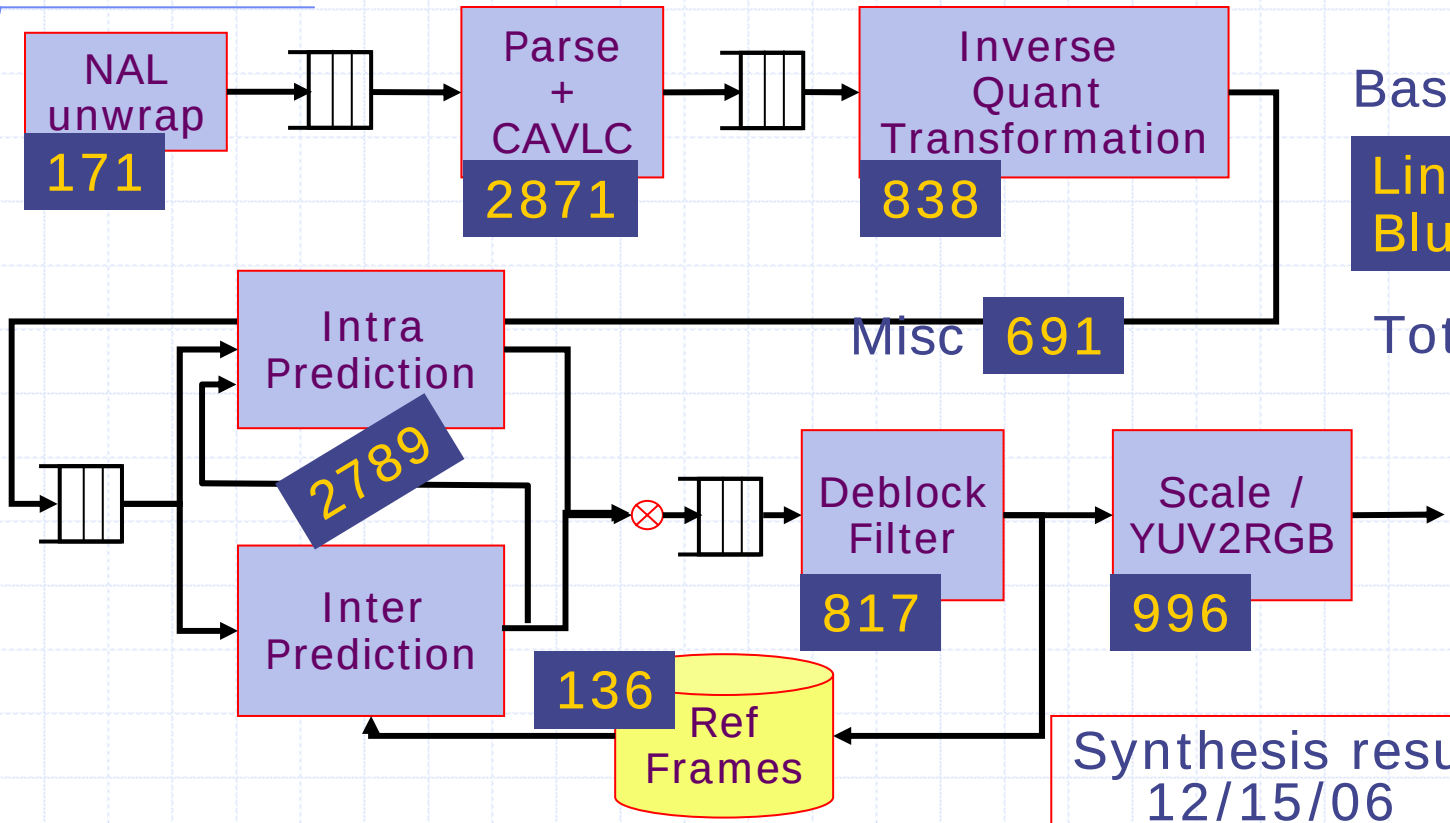
```
do_H264 :: Stream Chunk -> Stream Frame
do_H264 = let
    fMem :: IStructFrameMem MacroBlock
    fMem = makeIStructureMemory
    nalStream = nal inputStream
    parseStream = parse nalStream
    iqitStream = iqit parseStream
    interStream = inter iqitStream fMem
    intraStream = intra interStream
    deblockStream = deblock intraStream fMem
in
    deblockStream
```

Natural expression of all parallelism but very difficult to compile efficiently without domain specific information

Bluespec

```
module mkH264( IH264 )
// Instantiate the modules
    Nal nal <- mkNalUnwrap();
    ...
    DeblockFilter deblock <- mkDeblockFilter();
    FrameMemory frameB <- mkFrameMemoryBuffer();
//Connect the modules
    mkConnection(nal.out, parse.in);
    mkConnection(parse.out, iqit.in);
    ...
    mkConnection(deblock.mem_client, frameB.mem_writer);
    mkConnection(inter_pred.mem_client, frameB.mem_reader);
interface in = nal.in; //Input goes straight to NAL
interface out = deblock.out; // Output from deblock
endmodule
```

H.264 Decoder in Bluespec



Baseline profile

Lines of Bluespec

Total 9309

- Any module can be implemented in software
- Each module can be refined separately
- Behaviors of modules are composable
  - Good source code for multicores

Synthesis results 12/15/06

- Decodes 720p@75fps
- Critical path 50Mz
- Area 5.5 mm sq

Takeaway

- Parallel programming should be based on well defined modules and parallel composition of such modules
- Modules must embody a notion of resources, and consequently, sharing and time-multiplexed reuse
- Guarded Atomic Actions and Modules with guarded interfaces provide a solid foundation for doing so

Thanks