The Concurrency Challenge: Some inspirations from hardware design methods

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

The Concurrency Challenge, UW/MSR Workshop,
Semiahmoo Resort, WA
August 6, 2008
The power of numbers

- Last year 950M cell phones were sold as opposed to 100M PC
- India & Chine are selling ~7M new cell phones connection per month
  - In developing countries cell phone is the only computer most people have

A shift in research is underway from PCs to cell phone, not very different from the shift from Mainframes and Minis to PCs in early eighties.
The future would be dominated by the concerns of

cheap & powerful handheld devices

and

Powerful infrastructure needed to support services on these devices.
Current Cellphone Architecture

Two chips, each with an ARM general-purpose processor (GPP) and a DSP (TI OMAP 2420)

Many specialized complex blocks

Complex, high performance but must not dissipate more than 3 watts
Real power saving implies specialized hardware

- H.264 implementations in software vs hardware
  - the power/energy savings could be 100 to 1000 fold

*but our mind set is that hardware design is:*

- Difficult, risky
  - Increased time-to-market
- Inflexible, brittle, error prone...
  - How to deal with changing standards, errors

New design flows and tools can change this mind set
SoC & Multicore Convergence: more application specific blocks

- Application-specific processing units
- General-purpose processors
- Structured on-chip networks
- On-chip memory banks

Is consumer space different from enterprise space?
Server Microprocessors

- Also highly regular multicore with lots of specialized processing capability for
  - compression/decompression
  - encryption/decryption
  - intrusion detection and other security related solutions
  - Dealing with spam
  - Self diagnosing errors and masking them
  - ...

Server Multicore

more memory, more cores, ...

Quality-of-Service (QoS) aware networks on chips are essential for guaranteeing performance.
Software story?

Must think of the programming and architecture models together

- the resource requirements of programs should be “obvious”
- There should be an easy way of mapping programs on the underlying architecture

Smart allocation instead of virtualization of resources
New Parallel Software

*Synthesis* as opposed to *Decomposition*

A method of designing and connecting modules such that the functionality and performance are predictable

- Must facilitate natural descriptions of concurrent systems
- Modules must encapsulate many OS related services

A method of mapping such designs onto “multicores”

- Time multiplexing of resources complicates the problem – avoid time multiplexing as much as possible
- Resources which must be shared, e.g., networks, I/O, rely on QoS-aware allocations
Hardware-design inspiration
Hardware-design inspiration

- It is all about parallelism but there is no virtualization of resources
  - If one asks for two adder then one gets two adders – if one needs to do more than two additions at a time, the adders are time multiplexed explicitly

- Two-level compilation model
  - One can do a design with $n$ adders but at some stage of compilation $n$ must be specified (instantiated) to generate hardware. Each instantiation of $n$ results in different design

Analogy - In software one may want to instantiate a different code for different problem size or different machine configuration.
Multi-radio OFDM workbench

[MEMOCODE 2006, MEMOCODE 2007]
IP Reuse via parameterized modules

Example OFDM-based protocols

- MAC
- Scrambler
- FEC
- Interleaver
- Mapper
- Pilot & Guard Insertion
- IFFT
- CP Insertion
- De-Scrambler
- FEC Decoder
- De-Interleaver
- De-Mapper
- Channel Estimator
- FFT
- S/P
- Synchronizer
- TX
- RX
- Controller
- A/D
- D/A

- Standard specific potential reuse

- Different algorithms
- Different throughput requirements
- Reusable algorithm with different parameter settings
- 85% reusable code between WiFi and WiMAX
- From WiFi to WiMAX in 4 weeks
- Different algorithms

WiFi: 64pt @ 0.25MHz
WiMAX: 256pt @ 0.03MHz
WUSB: 128pt 8MHz

Convolutional
Reed-Solomon
Turbo

(Alfred) Man Chuek Ng,...
Video Codec: H.264

Chun-Chieh Lin (MIT MS thesis 2006)
Kermin Elliott Fleming [MEMOCODE 2008]
H.264 Video Decoder

Compressed Bits

NAL unwarp

Parse + CAVLC

Inverse Quant Transformation

Inter Prediction

Intra Prediction

Deblock Filter

Ref Frames

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

```c
void h264decode()
{
    int stage = S_NAL;
    while (!eof()){
        createdOutput = 0; stallFromInterPred = 0;
        case (stage){
            S_NAL: try_NAL();
            stage=(createdOutput) ? S_Parse:S_NAL; 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; 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

A (p)thread of each block

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

But there is no control over mapping

Processors:

- NAL thread
- Parse thread
- DeBlk thread
- Intrapr thread

Sleeping threads:

- IQ/IT thread
- Interpredict thread
StreamIT (Amarasinghe & Thies)  
*a more natural expression using filters*

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

Given the required rates StreamIT compiler can do a great job of generating efficient code
```

Feedback is Problematic!
Functional languages (pH)

```haskell
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 Hardware design language

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

Modularity and dataflow is obvious.
H.264 Decoder in Bluespec

Behaviors of modules are composable
Each module can be refined separately
Any module can be implemented in software

Decodes 1080p@70fps
Area 4.4 mm sq (180nm)

Compressed Bits

NAL unwrap

Parse + CAVLC

Inverse Quant Transformation

Inter Prediction

Intra Prediction

Deblock Filter

Frames

Baseline profile

Lines of Bluespec

Total 8137

Decodes 1080p@70fps
Area 4.4 mm sq (180nm)

Good source code for multicores
Current focus is on generating both hardware and software from Bluespec

Next Bluescript – a parallel scripting language
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.