criticalblue
Accelerating Embedded Software

Minimizing the cost and risk when migrating software to multicore platforms

24 April 2009

David Stewart | CEO

criticalblue
Accelerating Embedded Software

Agenda

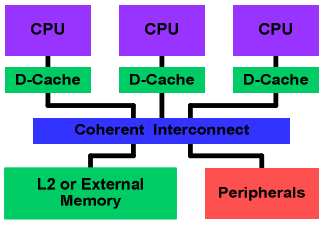
- Mainstream Multicore Programming
- What We Heard From Engineers
- How We Are Addressing These Issues

criticalblue
Accelerating Embedded Software

Multicore & Software: State of the Embedded Art (1)

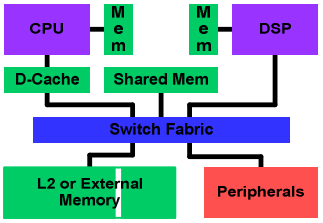
Desktop/Server/Telecom

Homogeneous
Shared memory
Coherent memory
eg x86 dual/quad
Sun UltraSparc T2



Embedded

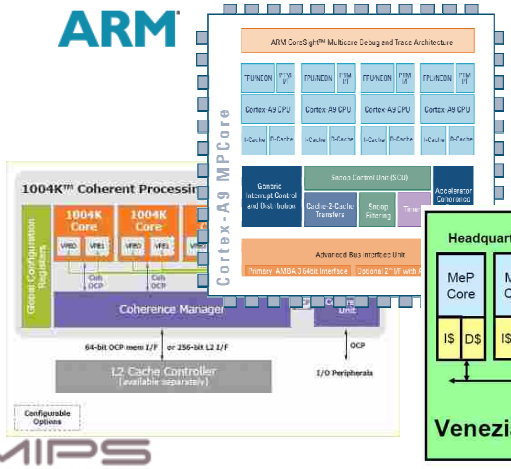
Heterogeneous
Distributed memory
Independent memory
eg TI OMAP
Most baseband ICs



criticalblue
Accelerating Embedded Software

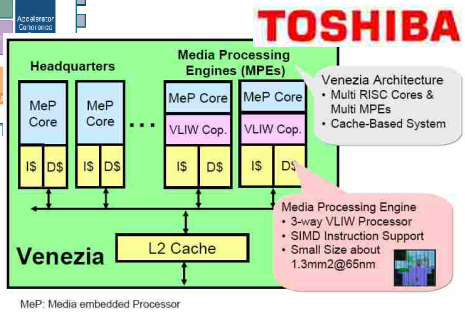
Multicore & Software: State of the Embedded Art (2)

ARM



- ***Regular C language***
- ***Threading API***
- ***1-4 cores***
- ***Shared memory***

TOSHIBA



MeP: Media embedded Processor

criticalblue
Accelerating Embedded Software

Multicore & Software: State of the Embedded Art (3)

- Putting lots of cores on silicon is relatively simple
- There are many clever memory and communication architectures
- However… the mass market of software developers haven't yet adopted multicore programming…
- Because…
- Software developers didn't ask for multicore!
- Software developers don't care about multicore block diagrams
- Software developers need motivation to migrate aggressively to multicore

criticalblue
Accelerating Embedded Software

Feedback From Engineers

- We love the C language
  - It's not that C is the perfect parallel language
  - 100s of millions of lines of existing code
  - Existing code maintenance is a huge overhead
- We love our Compiler/IDE/ISS
  - Companies typically standardize on a particular toolchain
  - Huge investment in the developers' desktop
  - Real world C/C++ is less portable than you might think
- We need motivation to move software to multicore
  - Benefits need to be clear before code changes are made
  - Multithreading simple to implement, hard to understand

criticalblue
Accelerating Embedded Software

Feedback From Engineers

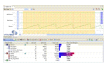
- **Do not synthesize my code!**
  - Maintaining sequential code is hard enough already
  - Requirement to retain control of source code changes
  - Identify minimal set of code changes to deliver benefit
- **We must have confidence in code changes**
  - Check that the code modifications deliver expected benefits
  - Check that no new problems were added
  - Check that the code is 'safe
- **We need education in multicore programming**
  - Lots of FUD, but also plenty of products in the market
  - Plenty of buzzwords, less clear explanation
  - How does this relate to my code?

criticalblue
Accelerating Embedded Software

Collating The Data

1. Analyze – characterize existing code. Identify parallelism determining issues

What might a practical and pragmatic flow look like?

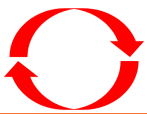


2. Explore – without code changes, evaluate alternative concurrent scenarios

3. Implement – select parallelization strategy and write code in existing IDE

4. Verify – check for code safety, race, threading and synchronization issues

5. Tune – inspect for further performance opportunities on target platform



criticalblue
Accelerating Embedded Software

JPEG Encoding

write header, image properties, and
decode and quantization tables

foreach **macroblock** {
convert rgb to yc_bc_r
transform to DCT frequency coefficients
quantize zig zag coefficients
encode huffman bitstream
}

criticalblue
Accelerating Embedded Software

Analysis 1

- **Understand serial code profile call tree**

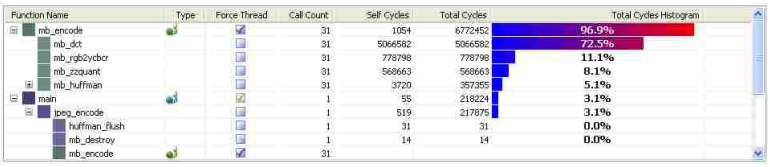
Function Name	Type	Force Thread	Call Count	Self Cycles	Total Cycles	Total Cycles Histogram
main			1	55	6990676	100.0%
jpeg_encode			1	519	6990327	100.0%
mb_encode			32	1081	6989763	100.0%
mb_dct			32	5232304	5232304	74.8%
mb_rgb2ycbcr			32	784485	784485	11.2%
mb_zzquant			32	586763	586763	8.4%
mb_huffman			32	3840	385130	5.5%
huffman_flush			1	31	31	0.0%
mb_destroy			1	14	14	0.0%

```
int mb_encode(mb_t *mb, image_t *image, int row, int col, huffman_t
*huffman) {
  int err;

  if ((err = mb_rgb2ycbcr(mb, image, row, col))) return err;
  if ((err = mb_dct(mb))) return err;
  if ((err = mb_zzquant(mb))) return err;
  if ((err = mb_huffman(mb, huffman))) return err;

  return 0;
}
```

criticalblue
Accelerating Embedded Software

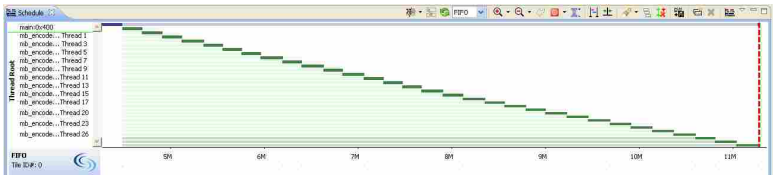
Exploration 1:1

- **Force mb_encode to run parallel**



Function Name	Type	Force Thread	Call Count	Self Cycles	Total Cycles	Total Cycles Histogram
mb_encode		☑	31	1054	6772452	96.9%
mb_dct		☐	31	5066582	5066582	72.5%
mb_rgb2ycbcr		☐	31	778798	778798	11.1%
mb_zzquant		☐	31	568663	568663	8.1%
mb_huffman		☐	31	3720	357355	5.1%
main		☑	1	55	218224	3.1%
jpeg_encode		☐	1	519	217875	3.1%
huffman_flush		☐	1	31	31	0.0%
mb_destroy		☐	1	14	14	0.0%
mb_encode		☑	31			

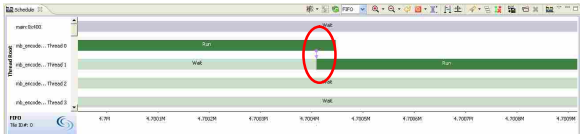
- **Resulting Schedule**



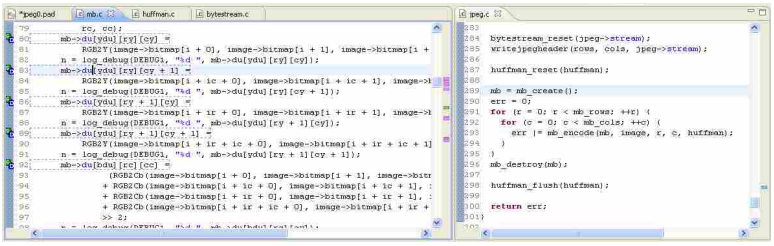
criticalblue
Accelerating Embedded Software

Exploration 1:2

- **Dependencies between macroblocks**



- **Source and Cause**



criticalblue
Accelerating Embedded Software

Coding 1

- **Thread the mb_encode function**

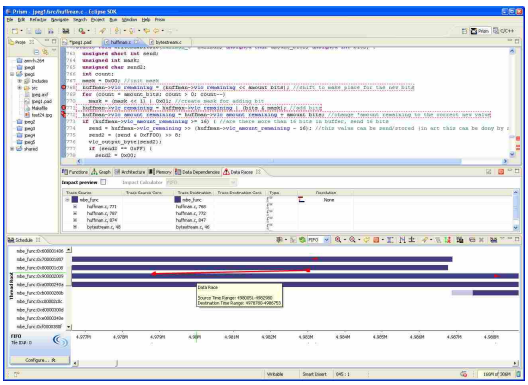
```
err = 0;
 for (r = 0, i = 0; r < mb_rows; ++r) {
   for (c = 0; c < mb_cols; ++c, ++i) {
     // pack each macroblock arguments and launch thread
     mbe_arg_pack(&arg[i], image, r, c, huffman, 0);
     // each mb created locally
     pthread_create(&thread[i], NULL, mbe_func, &arg[i]);
   }
 }

 // wait for threads to finish
 for (i = 0; i < mb_rows * mb_cols; ++i) {
   pthread_join(thread[i], (void *)&status);
   if (!err && arg[i].err) err = arg[i].err;
 }
```

- **"Advise, check, but don't touch our code!"**

criticalblue
Accelerating Embedded Software

Verification 1

- **Works on single core, 3.6/4x speedup**

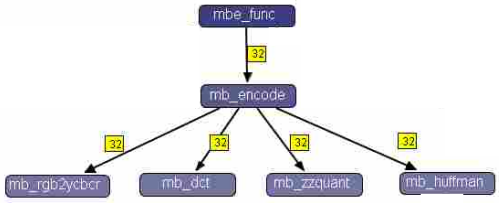


- **Fails on multicore, races in Huffman encoder**

criticalblue
Accelerating Embedded Software

Exploration 2

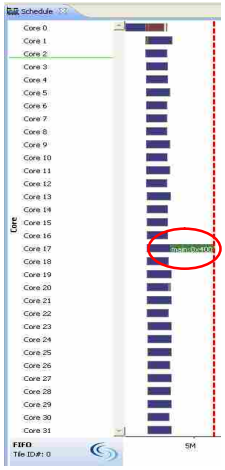
- Removing the macroblock ant-dependency is a typical parallel code optimization.
- Huffman has inter macroblock dependencies
- Move down a level and try running sub functions in parallel



criticalblue
Accelerating Embedded Software

Verification 2

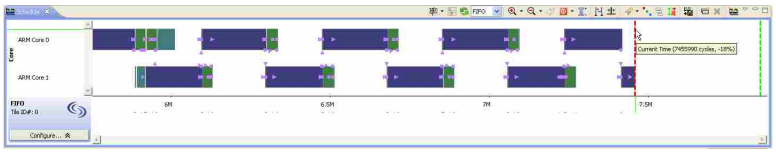
- Clean Schedule
- Most functions scale across cores
- Limited by serial Huffman tail
- Potentially uses more memory



criticalblue
Accelerating Embedded Software

Pass 3

- **Consider function pipelining**
  - Stage 1: rgb2ycbcr, dct
  - Stage 2: zzquant, huffman

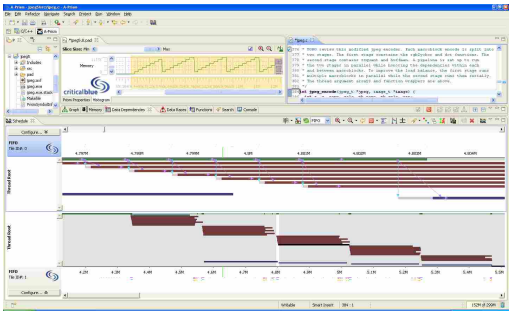


- **Poor load balancing, DCT dominates.**

criticalblue
Accelerating Embedded Software

Tuning 3

- **Balance pipeline work:**
  - Parallelize stage 1, serialize stage 2



- **Sensitive to DCT efficiency**

criticalblue
Accelerating Embedded Software

Summary

- We've made a start at a practical & pragmatic path for sequential to parallel software migration….
- Visit us at www.criticalblue.com and sign up for an evaluation….

Give us your feedback and help make Prism even better!