In the Days of IoT
Dealing with Software Parallelization for Heterogeneous Multicore Architectures

Yankin Tanurhan
Vice President R&D, Solutions Group

MPSoc, July 2014
There will be 25 billion devices connected to the Internet by 2015 and 50 billion by 2020.


© 2014 Synopsys, Inc. All rights reserved.
### IN A MINUTE

<table>
<thead>
<tr>
<th>Activity</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 million searches</td>
<td></td>
</tr>
<tr>
<td>571 new sites created</td>
<td></td>
</tr>
<tr>
<td>72 hours of video uploaded</td>
<td></td>
</tr>
<tr>
<td>350 gigabytes of data</td>
<td></td>
</tr>
<tr>
<td>204 million emails sent</td>
<td></td>
</tr>
<tr>
<td>104 thousand photos shared</td>
<td></td>
</tr>
<tr>
<td>11 thousand professional searches</td>
<td></td>
</tr>
<tr>
<td>278 thousand tweets</td>
<td></td>
</tr>
</tbody>
</table>

Source: QMEE, PC Mag.com, Go-Gulf.com, Business Insider, MailOnline.com, 4MAT, Intel (Jul 2013)
Internet of Things (IoT) – Fragmentation
Expect Many Types of Things; Highly Fragmented Market

By 2017, 50% of Internet of Things solutions will originate in startups less than three years old.

- Expect 10 billion shipments in 2020
- Many smart versions of existing product markets
- Key challenge: where to focus?

* Preliminary, September 2013
Complementary All-SoC Offering

**Processor IP**

- DDR PHY
- PCIe PHY
- HDMI PHY
- USB PHY
- 10G PHY
- SATA PHY
- MIPI D-PHY
- MIPI M-PHY

- DDR controller
- PCIe controller
- HDMI controller
- USB controller
- Ethernet controller
- SATA controller
- CSI-2, DSI controller
- UniPro, UFS, CSI-3, DigRFv4 controller

**Physical/Digital IP**

- AMBA 3 AXI & AMBA 2.0 AHB
- AMBA APB

- I2C
- GPIO
- UART

- Special Signal Processing
- ARC Audio Processor
- Video / Image ASIP Accel
- SD/MMC controller
- Embedded Memories (SRAM, ROM, NVM)

**ASIPs**

- ADCs
- DACs
- Audio Codecs
- Video Front End

**Datapath**

**Logic Libraries**

© 2014 Synopsys, Inc. All rights reserved. 8
Application Specific Processors (ASIP)
Optimizing for Specific Applications

General purpose
+ extension

RISC/DSP

ASIPs

Highly Specialized Instructions
Custom Registers and memories

Typically 10-100x performance compared to RISC

Typically 3-10x performance compared to RISC

ARC - APEX
XY/SIMD

Performance and Power Efficiency
“No MPSoC Design Without Tools”

• Tools at IP level (ASIP cores)
  – Architectural exploration
  – SDK generation: C compiler, ISS, debugger…
  – RTL generation

  → **IP Designer**

• Tools at IP subsystem level (multi-core)
  – Code parallelisation
  – Communication and synchronization
  – Multi-core platform generation

  → **MP Designer**
When is MP Designer Used?

• When the application is coded in sequential C code (for single-core execution), but parallelism must be introduced to improve performance or power
  – Targeting “multi-core” (≤ 10 cores) rather than “many-core” (e.g. ~100 cores)
    – Many-core is better served by languages with parallelism support, e.g. OpenCL
  – Application must allow for static parallelization (i.e. decided at design time)
  – Application must benefit from task-level parallelism (e.g. pipelined execution of tasks) and/or data-level parallelism* (parallelize loop iterations)

• Exploration is needed to achieve efficient load-balancing and low communication cost

• Option for specializing the individual core architectures
  – Bring IP Designer (ASIP design tool) in the loop
  – May result in heterogeneous multi-core architecture

• Option for generating a custom communication fabric
  – Using point-to-point connections

* Next release
MP Designer Tool-Suite

Typical users: multicore SoC design teams
System SW design teams
User-Guided Parallelization

Example: high-res JPEG encoding on 3-DLX architecture

C code

labels added for parallelization

Parallelization pragmas

pragmas referring to C labels

```
int main(int argc, char *argv[]) {
    init_all();
    parsection: {
        jpg_open: {
            jpg_fopen(JPG_filename);
            writeword(0xFFD8);  //SOI
            write_APP0info();
        }
        main_encoder(&in_img);
        jpg_close: {
            writeword(0xFFD9);  //EOI
            jpg_fclose();
        }
    }
    free(in_img.RGB_buffer);
    return 0;
}

void main_encoder(struct image* img) {
    vlc_init: {
        DCY=0;DCCb=0;DCCr=0;
    }
    for (ypos=0..height) {
        for (xpos=0....width) {
            for (blk=0..5) {
                SBYTE DU[64];
                loading:    
                    load_data_unit_from_RGB_buffer(img,
                                                xpos, ypos, Blk, DU);
                    process_DU(DU,blk);
                }
            }
        }
    }
    vlc_fini: {
            // Bit-alignment of EOI marker
            if (bytepos>=0) {
                writebits((1<<(bytepos+1))-1, bytepos+1);
            }
        }
...`
```
User-Guided Parallelization

• Exploring parallelization choices is easy and fast
  – Always work on sequential C code
  – Add source-code labels and parallelization pragmas
  – For each parallelization choice, MP Designer:
    – Checks all dataflow dependencies
    – Adds all required communication and synchronisation code, using a FIFO communication model
    – Provides feedback about performance and load balancing, memory and communication cost, data dependencies
FIFO Communication

- **Acquire/release interface** enables use of other processor’s data memory for storage of arrays (avoiding local copies)
- Synchronization implemented by polling on FIFO queue’s status (empty/full)
- Address translation for communicated pointers

```
// Producer side
DEFINE_SRC_FIFO(A,int,100)
void foo() {
    int* A;
    producer:
        A = FIFO_A_acq_put();
    for (...) {
        A[i] = ...;
        FIFO_A_rel_put();
    }
}

// Consumer side
DEFINE_DST_FIFO(A,int,100)
void foo() {
    int* A;
    consumer:
        A = FIFO_A_acq_get();
    for (...) {
        ... = A[i];
        FIFO_A_rel_get();
    }
}
```
Communication Fabric

• **Current**
  – Point-to-point links between communicating processors
  – Communication FIFOs mapped into destination processor’s local data-memory
  – Write conflicts resolved through local buffering
  – Address decoding logic
  – User constraints

• **Future**
  – Shared memory, shared bus

• **RTL & simulation model generated**
Exploration

Task Graph

- MP Designer generates a task graph for each parallelization alternative
  - Shows estimated processor loads
  - Shows data dependencies & communication cost

High-res JPEG encoding on 3-DLX architecture
Exploration

Task Graph

- Task graph for H263 encoding on 8 cores
  - Global dependency analysis automatically ensures correct communication & synchronisation
  - Manual design would be error-prone
Exploration

Activity Diagram

- MP Designer generates an activity diagram for each parallelization alternative
  - Dynamic view of core utilization

H263 encoding on 5-DLX architecture
## Exploration

- **JPEG encoding on multi-DLX architecture**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th># Cores</th>
<th>Parallelization</th>
<th>Mcycles*</th>
<th>Speed up</th>
<th>Load (%)</th>
<th>Efficiency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>1</td>
<td></td>
<td>7.1</td>
<td>1</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Original</td>
<td>2</td>
<td>ld+dct+q</td>
<td>vlcc</td>
<td>7.1</td>
<td>4.1</td>
<td>1.7</td>
</tr>
<tr>
<td>Original</td>
<td>3</td>
<td>ld</td>
<td>dct+q</td>
<td>vlcc</td>
<td>7.1</td>
<td>3.4</td>
</tr>
<tr>
<td>Original</td>
<td>4</td>
<td>ld</td>
<td>dct</td>
<td>q</td>
<td>vlcc</td>
<td>7.1</td>
</tr>
<tr>
<td>Optimised</td>
<td>2</td>
<td>ld+dct</td>
<td>q+vlcc</td>
<td>4.1</td>
<td>2.4</td>
<td>1.5</td>
</tr>
<tr>
<td>Optimised</td>
<td>3</td>
<td>ld</td>
<td>dct+q</td>
<td>vlcc</td>
<td>4.1</td>
<td>2.0</td>
</tr>
<tr>
<td>Optimised</td>
<td>3</td>
<td>ld</td>
<td>dct</td>
<td>q+vlcc</td>
<td>4.1</td>
<td>1.7</td>
</tr>
<tr>
<td>Optimised</td>
<td>4</td>
<td>ld</td>
<td>dct</td>
<td>q</td>
<td>vlcc</td>
<td>4.1</td>
</tr>
<tr>
<td>Split quant</td>
<td>3</td>
<td>ld</td>
<td>dct+q0</td>
<td>q1+vlcc</td>
<td>4.3</td>
<td>1.6</td>
</tr>
<tr>
<td>Dual load</td>
<td>5</td>
<td>ld0</td>
<td>ld1</td>
<td>dct</td>
<td>q</td>
<td>vlcc</td>
</tr>
</tbody>
</table>

* Cycles for 256x160-pixel image

- Entire exploration in only days of time
Summary

- IoT will drive many different MultiCore SoCs
- No (efficient) multicore SoC design without tools
  - Design and programming of individual ASIP cores
  - Multicore parallelisation and platform generation
- **MP Designer tool-suite**
  - Parallelisation from sequential C code
  - Exploration of functional parallelism
  - Static global dependency analysis
  - Efficient FIFO communication model (acquire/release interface)
  - All communication and synchronization code added automatically
  - Feedback about performance and load balancing, memory and communication cost, data dependencies
Thank You