Robots and Embedded Systems

MPSoC Programming In General

Kai Huang
What is MPSoC?

- Multi-Processor System-on-Chip
  - Multiple, usually heterogeneous, processing elements, a memory hierarchy, and I/O components, linked by an on-chip interconnect

- What is it used for?
  - Meet the performance needs of domain specific applications while limiting the power consumption

- What is MPSoC programming?
  - To develop applications for MPSoC platforms
What is MPSoC?

IBM CELL Broadband Engine

IBM

Cell Broadband Engine Processor

L2
(512KB)

Test & Debug Logic

Element Interconnect Bus

RAMBUS XDRAM Interface

Memory Controller

Power Processor Element

I/O Controller

Intel SCC
Why is MPSoC?

- Moors Law
- Power Wall
  - The Power Wall means faster computers get really hot.
Law of Physics: All electrical power consumed is eventually radiated as heat

Reasoning: use multiple cores with lower frequency to obtain the same overall performance
What Is the Problem of MPSoC?

- Memory Wall
- ILP Wall
- Power is still a issue
The Memory Wall means 1000 pins on a CPU package is way too many

- Bandwidth limit
- Distance between processing cores and memory

"Moore’s Law"
ILP Wall means a deeper instruction pipeline really means digging a deeper power hole

In-Order
Moderate-Pipe Superscalar/OOO
Very-Deep-Pipe Aggressive Superscalar/OOO

Made sense to go Superscalar/OOO: good ROI

Very little gain for substantial effort

Source: G. Loh
Power Wall for MPSoC

Reasoning: packing more transistors needs deeper sub micro CMOS techniques which results in larger leakage current
What Is the Problem of MPSoC?

- Memory Wall
- ILP Wall
- Power is still an issue

➤ How to program an MPSoC affects a lot!
Parallel Programming Success Stories

- Scientific/Super computing
  - Where data parallelism is abundant

- Data Centers
  - Independent queries
  - Few shared writes
  - Buying more hardware is cheaper and faster than efficient programming

- Computer Graphics
  - Near infinite, homogeneous parallelism
  - Portable libraries: OpenGL, DirectX
## Language Classification

<table>
<thead>
<tr>
<th>No Memory Concept</th>
<th>Complete Memory Abstraction</th>
<th>Garbage Collection</th>
<th>Stack Memory Abstraction</th>
<th>Stack Memory Abstraction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>malloc free</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Esteral**
- **Temporal logic**
- **Python**
- **Java**
- **Haskell**
- **C/C++**
- **Fortran**
- **OpenCL**
- **Assembly**
TIOBE Programming Community Index
Parallel Architectures in Two Dimensions

- Coarse Grain
  - Multi-Thread
  - Grid
  - GPU
  - Homogeneous

- Fine Grain
  - Stream
  - MPSoc
  - RISC
  - VLIW
  - SIMD
  - Heterogeneous

- Homogeneous
  - Vector
  - SIMD
  - Multi-Thread
  - Grid
  - GPU

- Heterogeneous
  - Stream
  - MPSoc
  - RISC
  - VLIW
  - SIMD
What We Need

- Coarse-grained level parallelism
- Design space exploration
- Easy to adopt Legacy code
- Non-functional prosperity verification
Distributed Operation Layer in ETHZ-TIK

- Application specification (XML & C)
  - Calibration data extraction and back-annotation
  - Functional simulation generation
    - Simulation on workstation
  - Mapping specification (XML)
    - System synthesis (HdS code generation)
      - Simulation on virtual platform/execution on physical platform
  - Architecture specification (XML)
    - Analysis model generation
      - Evaluation on performance data
Versatile MPSoC Software Design Flow

- Application (Process Network)
- MPSoC Architecture (PPC, interconnect, DSP)
- System specification
- Software synthesis
- Runtime environment
- Hardware platform

- DNA OS
- Linux + protothreads
- RTEMS
- DOL RTC software synthesis
- TIMA Hds generation tool (AGES)
- DOL functional simulation generation
- Mapping

- IBM Cell BE
- Atmel DIOPSIS
- Intel inside
- Single processor
Application Specification

Structure

- **Process Network**
  - Processes
  - SW channels (FIFO behavior)

- **Iterators**
  - Scalability for processes, SW channels, entire structures

Functional specification

- **Language:** C/C++
- **API:** DOL primitives

---

**Algorithm 1 Process Model**

1: `procedure INIT(DOLProcess p)`                       ▶ initialization
2:   initialize local data structures
3:   `end procedure`
4: `procedure FIRE(DOLProcess p)`                       ▶ execution
5:   DOL_read(INPUT, size, buf)                             ▶ blocking read
6:   manipulate
7:   DOL_write(OUTPUT, size, buf)                             ▶ blocking write
8:   `end procedure`
Scalability at Specification Level

- Separation of instruction/thread level parallelism (inside processes) and process-level parallelism.
- Use of iterators in
  - architecture specification
  - application specification
  - mapping specification
  \{(i, j) : 1 \leq i \leq N \land i \leq j \leq N\}
Definition of a process and Example

```c
// Process.h: definition of a process

// local information
typedef struct _local_states *LocalState;
// function pointer
typedef void (*ProcessInit)(struct _process*);
typedef int (*ProcessFire)(struct _process*);
// placeholder
typedef void *WPTR;

typedef struct _process {
    LocalState local;
    ProcessInit init;
    ProcessFire fire;
    WPTR wptr;
} Process;
...
```
Definition of a process and Example

// P2.h: header file
typedef struct _local_states {
    int index;    int len;
} P2_State;

#define PORT_IN 1
#define PORT_OUT 2

// P2.c: C file
void P2_INIT(Process *p) {
    p->local->index = 0;
    p->local->len = LENGTH;
}

int P2_FIRE(Process *p) {
    float i;
    if (p->local->index < p->local->len) {
        READ((void*)PORT_IN, &i, sizeof(float), p);
        WRITE((void*)PORT_OUT, &i, sizeof(float), p);
        p->local->index++;
    }
    if (p->local->index >= p->local->len)
        DETACH(p);

    return 0;
}

// main.c: sample main function
int main() {
    Process p2; P2_State p2_state;
    p2.local = p2_state; p2.init = p2_INIT; p2.fire = p2_FIRE;
}
Scalable MPEG-2 Decoder

<!— N1 is the number of gops processed in parallel ——>
<variable name="N1" value="1" />
<!— N2 is the number of macroblocks processed in parallel ——>
<variable name="N2" value="4" />
<!— N3 is the number of blocks processed in parallel ——>
<variable name="N3" value="2" />

<!— instantiate processes ——>
<process name="dispatch_gops">
  <iterator variable="i" range="N1">
    <port type="output" name="out" append function="i" />
  </iterator>
  <source type="c" location="dispatch_gops.c" />
</process>
...

<iterator variable="i" range="N1">
  <iterator variable="j" range="N2">
    <process name="dispatch_blocks">
      <append function="i" />
      <append function="j" />
      <iterator variable="k" range="N3">
        <port type="output" name="out" append function="k" />
      </iterator>
    </process>
  </iterator>
  ...

...
Scalable MPEG-2 Decoder

$N_1 = 1, \ N_2 = 4, \ N_3 = 2$
Target Platform Abstraction (1)

- **RISC 0**
- **DSP 0**
- **MEM**
- **RISC 1**
- **DSP 1**

- **C_inter_tile**

**Symbols:**
- **processor**
- **memory**
- **hardware channel**
- **input / output / bidirectional port**
- **connection**
Target Platform Abstraction (2)

- `<processor name="processor1" type="DSP">
  <port name="processor_port" type="duplex"/>
  <configuration name="clock" value="100 MHz"/>
</processor>

+ `<processor name="processor2" type="RISC">
+ `<memory name="sharedmemory" type="DXM">

- `<hw_channel name="in_tile_link" type="bus">
  <port name="port1" type="duplex"/>
  <port name="port2" type="duplex"/>
  <port name="port3" type="duplex"/>
  <configuration name="buswidth" value="32bit"/>
</hw_channel>

- `<connection name="processor1link">
  <origin name="processor1">
    <port name="processor_port"/>
  </origin>
  <target name="in_tile_link">
    <port name="port1"/>
  </target>
</connection>

+ `<connection name="processor2link">
+ `<connection name="memorylink"">
Mapping Specification

- `<binding name="generator_binding" type="computation">
  <origin name="generator" />
  <target>
    <resource name="processor2" />
  </target>
  <schedule type="roundrobin" />
</binding>

+ `<binding name="consumer_binding" type="computation">
- `<binding name="square_binding" type="computation">
  <origin name="square" />
  <target>
    <resource name="processor1" />
  </target>
  <schedule type="fixedpriority">
    <configuration name="priority" value="1" />
  </schedule>
</binding>

+ `<binding name="C1_binding" type="communication">
+ `<binding name="C2_binding" type="communication">

Reasoning: Shift the design to system level and limit the ILP Wall to individual cores

Iterated mapping possible
DOL Design Flow Demonstration

XML visualization

DOL Design
Flow Demonstration

application developer

XML to HdS refinement

network XML

C source code

functional simulation instr.

accurate simulation

mapping design space exploration

application developer

network XML

architecture

DOTTY

control

src_1

fifo_src_cconv_1_1

fifo_src_cconv_1_0

fifo_src_cconv_1_2

fifo_src_cconv_1_3

convolver_wrapper_1_0

convolver_wrapper_1_1

convolver_wrapper_1_2

convolver_wrapper_1_3

fifo_conv_sum_1_0

fifo_conv_sum_1_1

fifo_conv_sum_1_2

fifo_conv_sum_1_3

fifo_conv_sum_2_0

fifo_conv_sum_2_1

fifo_conv_sum_2_2

fifo_conv_sum_2_3

fifo_conv_sum_3_0

fifo_conv_sum_3_1

fifo_conv_sum_3_2

fifo_conv_sum_3_3

src_2

convolver_wrapper_2_0

convolver_wrapper_2_1

convolver_wrapper_2_2

convolver_wrapper_2_3

fifo_conv_sum_2_0

fifo_conv_sum_2_1

fifo_conv_sum_2_2

fifo_conv_sum_2_3

fifo_conv_sum_3_0

fifo_conv_sum_3_1

fifo_conv_sum_3_2

fifo_conv_sum_3_3

src_3

convolver_wrapper_3_0

convolver_wrapper_3_1

convolver_wrapper_3_2

convolver_wrapper_3_3

fifo_conv_sum_3_0

fifo_conv_sum_3_1

fifo_conv_sum_3_2

fifo_conv_sum_3_3
DOL Design Flow Demonstration

application developer

to HdS refinement

...
DOL Design Flow Demonstration

- process network XML
- C source code
- distributed operation layer
- XML visualization
- functional simulation
- instr.-accurate simulation
- ann. process network XML
- mapping design space exploration
- mapping XML
- architecture XML
- application developer
- to Hds refinement
DOL Design Flow Demonstration

- XML visualization
- Functional simulation
- Instr. accurate simulation
- Network XML mapping
- Design space exploration
- Application developer to Hds refinement
- Distributed operation layer
- C source code functional simulation
- Network XML
- Application developer
- Architecture XML
- Application developer to Hds refinement
Summary

- MPSoC programming
  - Reasons
  - Challenges
  - Principles
  - Practices