Delivering Application Portability for ARM Processor-based Heterogeneous Multicore SoCs

Frank Schirrmeister
Imperas, Inc.
Agenda

- Introduction – The End of the Single Core Era
- Time Warp Forward – Application Optimization Case Study
- The Path to MP – Software Migration
  - Migration Examples & Challenges
  - MPSoC Software Development Environments
    - Characteristics
    - Requirements
    - Potential Solutions
- Summary
The End of the Single Core Era

**Processor Performance**
- Is no longer growing
- Parallelism was addressed in hardware using instruction level parallelism

**Instruction Level Parallelism**
- Runs out of steam
- Issuing more than four instructions in parallel has little effect on most applications

**Power Ceiling**
- Has essentially stopped progress in conventional processor core development

Source: Intel, Imperas

ARM Developers’ Conference & Design Pavilion 2007
The Business Breakdown

Proposed approaches:
- Faster processors?
  - Limited by power!
- Dedicated hardware accelerators?
  - Do not address flexibility to optimize Time in Market!

“Von Neumann is a poor use of scaling – all the energy is going on the communication between the processor and the memory. It’s much better to use 20 microprocessors running at 100MHz than one at 2GHz”

Hugo de Man
IMEC

Multiprocessor Systems on Chips (MPSoCs) are the only feasible alternative …
MPSoCs are the Solution

- Multiple CPUs
  - Custom
  - 3rd Party IP
- Heterogeneous
  - Homogeneous for very specific applications
- Complex communication

**Main Challenge: Software**
- Complexity explodes
- Sequential software on individual cores fails
- Multiple processors present various challenges
- Partitioning
  - Parallelization
  - Optimization
MPSoC Programming Challenges

1. How do I **deliver** the software and programming environment to **customers**?

2. How do I build a **programming environment**?

3. Which **programming model and methodology** do I adopt for my customers…?

4. How do I **parallelize** the application?

5. Can I **automate** the **software verification**?

6. How do I develop and **verify** a range of reference applications for an MPSoC?

Which CPU do I choose?

Is this the right bus architecture?

Which hardware IP do I need to develop?

How do I model my custom CPU?

5. How do I develop and **verify** a range of reference applications for an MPSoC?
Time Warp
How the Future Could Look Like

Application Optimization Case Study
The Application: Picture in Picture Video

**Requirement:** Image processing has to be done within 1/30s of a second to avoid picture misses for video display

**Design Objective:** Optimize application on target MPSoC platform via re-mapping of tasks – try to increase slack in the system!
Target MPSoC Description

- Multiple CPUs
- Heterogeneous – 2 ARM, 3 DSPs
- Bus hierarchies
- Complex communication
- Memories etc.
Application Characterization

- Number of instructions
- Number of memory accesses
- Latency
- Communication
- Buffer depth
- Impact of processor configurations
Optimizing Applications on MPSoC Platform
Application Optimization!
Re-Mapping Optimization Results

Results

✓ 60%+ reduction in application execution time just by remapping
✓ Resulting slack in system allows users to
  ▪ Reduce clock frequency or
  ▪ Add more functionality
Optimization Results with Re-Mapping

*Interesting Result: Four processor system performs surprisingly well*

*Different times due to communication overhead*
One Step More: Optimization Across Platforms

Software Optimization

Optimizing applications on different platforms

Application Execution Time

Communication Overhead

ARM Developers’ Conference & Design Pavilion 2007
The Dirty Reality

How do I ready my software for MP?

Software Migration to MP is Hard!
MPSoC Design & Programming Methodology

1. Parallelize
   - Use sequential / legacy code
   - Identify parallelization potential
   - Explicitly articulate parallelization and communication
   - Debug & verify parallelized application against legacy reference

2. Map to specific platform
   - Define design goals
   - Optimize resources
   - Optimize Communication
   - Deploy application
   - Verify application

3. Optimize application
   - Define design goals
   - Re-map application across resources
   - Debug and verify application against legacy reference
MPEG-2 Decoder MP Porting Example

- Code issues to be identified to find and extract:
  1. Who uses which data when?
  2. Who is dependent on whom, i.e. allowing pipelining?
  3. Which tasks are independent and can be fully done parallel?
  4. Where are the massively data parallel memory accesses coming from?

- Users need full visibility for
  - Code execution and coverage
  - Memory and register usage
  - Locks, Threads …
MPEG-4 Decoder MP Porting Example

- Code issues to be identified to find and extract:
  1. Must identify the blocks that can run in parallel
  2. Identify and eliminate data dependencies
  3. Define memory organizations between blocks that optimize the information transfer
  4. Verify against reference to ensure correctness

- Users need
  - Full visibility into code issues
  - Understand memory
  - Simulation for verification
**Application Coding**

- Finding Potential Coding Issues
  - Global and static variables are potential pitfalls in their usage
  - Difficult to determine whether real issues exist without simulation and analysis.

- Re-Architecture of Code
  - Re-architected code had all global data placed into dynamically allocated structures, each one unique to a particular part of the parallel execution phase.
  - This ensured that individual parallel processes would not ‘stamp’ on each others data during encoding.
  - Static variables in state holding functions had to be extracted, initialized and iteratively passed back.
  - This would ensure that distinct parallel tasks did not rely upon the state of these variables.

- Optimization
  - Needs some method of being able to identify how much time is being spent in each processor executing code (and where in the code it is spending its time).
  - One needs to identify where sections of code are spending time waiting to access resources, and more importantly what is causing this code to wait.

“During the porting of the application from a single to multi processor environment, I would estimate that 2/3 of the time was spent tracking down obscure and difficult to find bugs.”

Video Software Developer

Source: [happy.emu.id.au/lab/tut/dttb/dtbtut3b.htm](http://happy.emu.id.au/lab/tut/dttb/dtbtut3b.htm)
Coding Style

- Global variables
  - Sequencing of global variables deterministic in a single processor system
  - Ordering differences in a multi-processor system
  - Any global variable must be protected using a semaphore
  - Find all global and local variables
  - 'Package' global variables into unique structures which can be passed safely from processor to processor during different encoding phases

- Function Static Variables
  - Static variables cause problems in an MP environment
    - functions calls from multiple processors cause the state to be corrupted
  - Example:
    - VLC is called many times whilst encoding a packet of data, and local static variables are used to store state regarding the alignment of bits, and storage of bits which overflow from one byte to the next.
    - Due to many processors calling this function, the state is corrupted from call to call.
  - Maintain the state external to the function and pass it back each time.
  - Ensure all function static variables are local to the process carefully handling of the overlapping memory map

Shared Data Format vs. Processor Data Types

- Passing complex data structures between heterogeneous processors.
- Compilers may pack data in different ways.
- If data is packed in one fashion in a sending processor, we must be certain that it can be unpacked in the same manner on a receiving processor.

Memory Protection & Semaphores

- Static creation between processors
  - compile time management
  - When ‘n’ processors are intending to share a variable, there must be some resolution about where that variable resides, and how its semaphore will be initialized.
  - One cannot assume that its semaphore will initialize in the correct state, therefore something needs to perform this initial action.

- Dynamic creation between processors
  - run time management
  - When shared variables are created dynamically, the processor which creates the variable needs a method to communicate where it resides to other processors intending to access this variable.
  - This communication itself must occur through communication channels
MPEG-2 Encoder MP Porting Example 3/3

**Debugging**
- Debugging Control flow
  - Scope of what one wants to step through is very important
  - Traditional solution:
    - either step into (step),
    - or step over (next) each line of code.
  - MP environment
    - many lines of code are executing concurrently, needs more control to decide what we see next.
    - User needs to be able to specify which process(es) he is interested in debugging and being able to step/next the debugger into and/or across each process. In the simplest case users will want to maintain the focus in a single process. In a more complicated scenario, they may want the focus to cover either a selected list, or a global set of processes.
- Debugging Data flow
  - Sometimes it is more important to track the flow of data through a system
  - Example MPEG-2 Encoder
    - Packet of data is initialized by one processor,
    - encoded by another processor,
    - and finally merged by yet another processor.
- Debugging Process Waiting / Running, Race Detection / Deadlock / Starvation
  - Need to be able to identify which processes are running and awaiting access to some resource.
- Host system availability & Testbench monitor
  - Get data in and out of a system
- Memory Leak Checking
  - In an MP system memory can be allocated and de-allocated in different processors
  - Malloc/free routines may be custom routines
  - Need flexibility to watch custom and pre-defined routines with temporal assertions
Summary: Getting to Parallel Code

1. Parallelize
   - Use sequential / legacy code
   - Identify parallelization potential
   - Explicitly articulate parallelization and communication
   - Debug & verify parallelized application against legacy reference

2. User Needs:
   - Simulation & debug with deep insight
     - Coverage
     - Memory
     - Execution time
     - Dependencies
     - Assertions
   - Once modifications are done new code needs to be verified against golden reference
     - Simulation
     - Advanced verification

Without **MP Simulation, MP Debug** and **MP Verification** porting to MP Systems is not possible
Multi Processor System on Chip
Software Development Environments
Requirements: Simulation

- Today, simulation speed is limiting
- Need faster simulation
  - Enabling trade offs
  - Flexibility: appropriate accuracy at appropriate speed

Source: ARM IQ Magazine

Required Speed

- Application Software Development
- Firmware Development
- Driver Development
- Transaction Level Verification (SystemC)
- Hardware / Software Co-Verification
- RTL Hardware Verification

ARM Developers’ Conference & Design Pavilion 2007
Today’s Approaches with SystemC

**SW development**
- Run application code compiled for host
  - Fast
  - Not instruction accurate
  - May give different results
- Model peripherals and communication in SystemC
  - Special OS code
  - Not timing accurate
  - Performance bottleneck

**SW verification**
- Run application code on ISS
  - Slow
  - Instruction accurate or cycle accurate
  - May use vendor debugger
- Wrap ISS in SystemC
  - Memory inside or outside
  - Speed or accuracy
- Model peripherals and communication in SystemC
  - OS can run on ISS

SystemC inappropriate beyond verification …
Often too much detail, only reaches single digit MIPS, triple digit MIPS required
Other Approaches …

**Code Morphing**
- Run application code compiled for ISS but translated into host instructions
  - Fast
  - Instruction accurate
  - May use vendor debugger

**Hardware**
- FPGA Development systems
  - Fast
  - Late in the flow … a fair amount of implementation has to be done
- Emulation
  - Pretty fast …
  - Sometimes painful to set up (order of weeks)
  - Also late in the flow
Requirements: Solve MP Nightmares

**Data Races**
Leading to unpredictable results

- Write SystemStatus_{new}
- Write UserRequest_{new}
- Read SystemStatus_{old}
- Read UserRequest_{new}
- ChangeSystemState

**Deadlocks**
Bringing the MP application to a halt...

- Lock frame F2 to write
- Try to lock Frame F1 to write
- Release lock on frame F1
- Release lock frame F2
- Lock frame F1 to write
- Try to lock frame F2 to write

**Memory Corruption**
Leading to crashes & inconsistent results

- malloc (sec 1)
- malloc (sec 2)
- free (sec 1)
- malloc (sec 3)
- free (sec 3)

**Stalls**
Locks causing performance issues

- Lock Frame F1
- Lock Attempt
- Lock Frame F1
- Lock Attempt
- Stalls

---

**Multicore Design Simplified**

**Imperas**

**ARM Developers’ Conference & Design Pavilion 2007**
MP Breaks Debugging on Hardware

- MP debug issues break hardware based debug!
  - Memory?
  - Complex Breakpoints?
  - Control and roll back?
  - Synchronization of processors?
  - Hold all processors except one?
  - Access to registers?
  - Swap state?
  - Defect reproduction?

- Also:
  - Late
  - Fixed
  - ...

- Needs virtual representation with full control and observability!

http://shemesh.larc.nasa.gov/fm/spider/

ARM Developers’ Conference & Design Pavilion 2007
Requirements: Debugging

- Single core debugging approaches don’t scale to MPSoC
- Multiple confusing, independent sessions
- Different debuggers for different processors
- Unaware of full system context
- Cross processor debug hard if not impossible

➤ Need true multi processor debug on virtual representation of the chip

No wonder that more than half of current embedded design projects are behind schedule …
Requirements: Programming

Today
- Various languages
  - Do not express parallelism
  - Often limited to specific application domains
- Several incompatible programming models used for special applications
  - OpenMP
  - YAPI
  - DSOC
  - SMP
  - xUML
  - ...

Long Term MPSoC Requirements
- Need a model to “migrate” to
  - Including automated path to implementation
- Appropriate Programming Models
  - Task level parallelism
  - Flexibility
  - Efficiency
One Approach …

- Communication structure is separated from tasks
  - Coordination language
- Various modes of communication can be supported
  - blocking, non-blocking
- Communication can be implemented in various ways
  - Depending on the platform
Requirements: Software & Automation

**Today**
- Limited SW support
  - SystemC models slow for SW developers.
  - Models not well verified
  - Debugger integration poor
- Focus on Analysis
  - “here you go … now fix it yourself manually and re-simulate”

**Long Term MPSoC Requirements**
- True HW/SW Interaction
  - Higher levels of speed/accuracy trade-off
  - Easily verifiable models
- True HW/SW Automation
  - SW Mapping & Optimization
  - HW/SW Optimization
  - “here is the solution for your power/performance objectives”
Programming Environment Requirements

Application Programming model handling concurrency and communication abstractions using e.g. Threads(?), C API, platform independent, tools/automation to map to MP hardware

Virtual MPSoC with Programming Environment

MP Application Programming model,
Tools & Methodology
MP Environment
MP Verification & Debug
MP Simulation
Models (User Modeling & Model Library)
Model Library

Integrated Development Environment - Eclipse
customizable and extendable, providing control, scripting and analysis

Packaged Delivery to programmers

Multi-Processor Debug

Automated Software Verification

Multi-Processor Simulation in the 100 MIPS speed domain
of platforms including multiple processors, memory hierarchies, communication, peripherals, and test environment

Modeling open to, and done by users
CPUs, Sub-systems, Peripherals, MPSoC Platforms and Testbenches

ARM Developers’ Conference & Design Pavilion 2007
Summary

- The Single Core Era is over
- Hardware reacts with shift to MPSoCs
- With that the challenges move into MP programming
- MP programming causes nightmares
  - Porting, 1st Migration from serial to parallel
  - Simulation, debugging and verification
- Existing sequential approaches will fail
- New MPSoC Programming Environments are needed to enable the next generation hardware/software devices
  - Short term: MP Modeling, Simulation, Debug, Verification
  - Mid term: Programming Models to port & migrate to
  - Long term: Automation of Application to MPSoC Mapping