VIPPE: Parallel simulation and performance analysis of complex embedded systems

L. Diaz, E. Gonzalez, E. Villar, P. Sanchez
University of Cantabria

Motivation

- The MPSoC
  - Multi-processing platform
    - ASIC
    - FPGA
    - Commercial multi-processing platform
  
- SW-centric design methodology
  - Most of the functionality implemented as Embedded SW
  - With ‘some’ application-specific HW

- Functional and extra-functional verification
  - Simulation & Performance analysis
  - Virtual HW/SW platform
  - SW integration
State of the art

- **Functional Simulation**
  - Fast but inaccurate

- **Native Simulation**
  - Fast and accurate enough

- **ISS**
  - Accurate but (even JITC) too slow

- **RTL**
  - extremely slow
  - only valid for final verification

---

SW Simulation Technologies

- **Embedded SW simulation**

  - Functional Simulation
  - Native & Trace-based code simulation
  - Fast Computation & Communication estimations
  - Native & Trace-based co-simulation
  - Accurate Computation & Communication estimations

  - Virtual Models
    - ISS Discrete-Time Models
    - HDL Discrete-Event Models

  - Executable Specification
  - Co-Design

  - UML/MARTE
    - MDA

  - SystemC

  - HW Platform

  - Embedded SW

  - C

  - Compilation

  - VHDL
    - Verilog

  - HW/SW Implementation

  - Requirements

  - HW/Reuse

  - Behavioral Synthesis

  - RTL Synthesis
Native simulation methodology

- Based on basic blocks

VIPPE annotation process

- Native simulation
VIPPE Kernel

- Each target thread modeled as a host thread
- Target RTOS as an additional thread
- No need for thread locking
- Threads locking
  - shared variable
  - synchronization element
    • i.e. semaphore

VIPPE Kernel

- Informing the kernel thread about their estimations of individual thread times, access to memory, etc. via shared variables.
VIPPE API

- Optimal reduced set of primitives
  - Kernel services

- OS API implementation

<table>
<thead>
<tr>
<th>MANAGEMENT OF PROCESSES</th>
<th>MANAGEMENT OF TIMES</th>
</tr>
</thead>
<tbody>
<tr>
<td>process_create</td>
<td>time_real_watch</td>
</tr>
<tr>
<td></td>
<td>time_user_watch</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MANAGEMENT OF THREADS</th>
<th>MANAGEMENT OF SEMAPHORES</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_prio</td>
<td>semaphore_create</td>
</tr>
<tr>
<td>set_prio</td>
<td>semaphore_wait</td>
</tr>
<tr>
<td>get_id</td>
<td>semaphore_post</td>
</tr>
<tr>
<td>thread_create</td>
<td>semaphore_watch</td>
</tr>
<tr>
<td>thread_exit</td>
<td></td>
</tr>
<tr>
<td>thread_delete</td>
<td></td>
</tr>
<tr>
<td>thread_wait_for_end</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MANAGEMENT OF SIGNALS</th>
<th>MANAGEMENT OF AFFINITIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>vippe_signal</td>
<td>uc_PE_mapping</td>
</tr>
<tr>
<td>vippe_kill</td>
<td>uc_PE_dismapp</td>
</tr>
<tr>
<td></td>
<td>uc_take_OS_id</td>
</tr>
</tbody>
</table>
POSIX API

- POSIX API
  - Events
  - Concurrency
  - Synchronization
  - Timing and I/O

| Events            | vippe_signal  
vippe_kill        |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrency</td>
<td>process_create</td>
</tr>
</tbody>
</table>
                          | thread_create  
                          | thread_delete  
                          | thread_wait_for_end  
                          | get_id           
                          | get_prio         
                          | set_prio         |
| Synchronization   | semaphore_create  
                          | semaphore_wait  
                          | semaphore_post  
                          | semaphore_watch  |
| Timing            | time_real_watch  
                          | time_user_watch  |
| I/O               | semaphore_wait  
                          | semaphore_post  |
POSIX API annotation

• POSIX API
  – Annotated in the same way as user’s code

• VIPPE API
  – Annotated at function-level
  – Execution time and power depending on target processor

Experimental results

• Evaluated with the PARSEC benchmark suite
  – Princeton Application Repository for Shared-Memory Computers

• Host
  – 8 Intel Xeon E5-2687W at 3.10GHz
  – Each processor has 8 cores, thus the host platform integrates 64 cores with SMP capability
  – 64Gb of RAM and 20Mb of cache
### Speed-up vs number of target threads

- The number of cores of the target platform has been limited to 4.
- Although the native simulation requires more host time than the original benchmark, the speed-up is similar.
- Maximum difference between original code and simulation is about 15%.

![Graph showing speed-up vs number of target threads](image)

### Execution time vs number of cores in the target platform

- Performance improvement in simulation execution time is independent of the number of processors in the target platform.
- Depends only on the number of host processors and application threads.

![Graph showing execution time vs number of cores](image)
Impact of the number of host cores

- When the number of threads is higher than the number of host processors, the time penalty due to thread creation, scheduling time (i.e. context switching), etc. increases execution time.

- When the number of threads is equal to or lower than the number of host processors, it can be seen that the execution time of the simulation is reduced proportionally to the number of threads in the application.
Conclusions

- VIPPE: Native simulation tool
  - Efficient
  - Parallel
- Simulation of application SW
  - multi-processing platforms
- VIPPE API
  - optimized, general-purpose
  - supports complex OS APIs (i.e. POSIX)
- Experimental results
  - advantages in SW simulation on multi-core workstations